Specification and Verification using Temporal Logics

Stéphane Demri, Paul Gastin

Chapter 1

Specification and Verification using Temporal Logics*

Stéphane Demri & Paul Gastin
LSV, ENS Cachan, CNRS, INRIA Saclay, France
{*demri,gastin*}*@lsv.ens-cachan.fr*

This chapter illustrates two aspects of automata theory related to linear-time temporal logic LTL used for the verification of computer systems. First, we present a translation from LTL formulae to Büchi automata. The aim is to design an elementary translation which is reasonably efficient and produces small automata so that it can be easily taught and used by hand on real examples. Our translation is in the spirit of the classical tableau constructions but is optimized in several ways. Secondly, we recall how temporal operators can be defined from regular languages and we explain why adding even a single operator definable by a context-free language can lead to undecidability.

Keywords: temporal logic, model-checking, Büchi automaton, temporal operator, context-free language

1.1. Introduction

Temporal logics as specification languages. Temporal logics (TL) are modal logics [1] designed to specify temporal relations between events occurring over time. They first appear as a branch of logic dedicated to reasoning about time, see e.g. [2]. The introduction of TL for reasoning about program behaviours is due to [3]. Among the desirable properties of formal specification languages, the temporal logics have an underlying flow of time in its models, define mathematically the correctness of computer systems, express properties without ambiguity and are useful for carrying out formal proofs. Moreover, compared with the mathematical formulae, temporal logic notation is often clearer and simpler. This is a popular formalism to express properties for various types of systems (concurrent programs, operating systems, network communication protocols, programs with pointers, etc.). An early success of the use of TL has been the verification of finite-state programs with TL specifications, see e.g. [4, 5].

*This work has been partially supported by projects ARCUS Île de France-Inde, ANR-06-SETIN-003 DOTS, and P2R MODISTE-COVER/Timed-DISCOVERI.

Automata-based approach. Model-checking [4, 6] is one of the most used methods for checking temporal properties of computer systems. However, there are different ways to develop theory of model-checking and one of them is dedicated to the construction of automata from temporal logic formulae. In that way, an instance of a model-checking problem is reduced to a nonemptiness check of some automaton, typically recognizing infinite words [7]. This refines the automata-based approach developed by R. Büchi for the monadic second-order theory of $\langle \mathbb{N}, < \rangle$ in [8] (for which nonelementary bounds are obtained if the translation is applied directly). This approach has been successfully developed in [9] for linear-time temporal logics and recent developements for branching-time temporal logics with best complexity upper bounds can be found in [10], see also a similar approach for description logics, program logics or modal logics [1].

On the difficulty of presenting simple automata The translations from temporal formulae into automata providing best complexity upper bounds, often require on-the-fly algorithms and are not always extremely intuitive. For instance, on-the-fly algorithms for turning specifications into automata and doing the emptiness check can be found in [11–13]. In order to explain the principle of such translations, it is essential to be able to show how to build simple automata for simple formulae. For instance, the temporal formula $\mathsf{X}^n\, p$ stating that the propositional variable p holds at the n-th next step, may lead to an exponential-size automaton in n when maximally consistent sets of formulae are states of the automata even though $\mathsf{X}^n\, p$ has a linear-size automaton. This gain in simplification is of course crucial for practical purposes but it is also important to have simple constructions that can be easily taught. That is why we share the pedagogical motivations from [14] and we believe that it is essential to be able to present automata constructions that produce simple automata from simple formulae.

Our contribution. This chapter presents two aspects of automata theory for LTL model-checking and it can be viewed as a follow-up to [9, 15, 16]. First, we present a translation from LTL formulae to generalized Büchi automata such that simple formulae produce simple automata. We believe this translation can be easily taught and used by hand on real examples. So, Section 1.2 recalls standard definitions about the temporal logics LTL and CTL* whereas Section 1.3 provides the core of the translation in a self-contained manner. A nice feature of the construction is the use of target transition-based Büchi automata (BA) which allows us to obtain concise automata: the acceptance condition is a conjunction of constraints stating that some transitions are repeated infinitely often. This type of acceptance condition has already been advocated in [12, 14, 17–20]. Secondly, we consider richer temporal logics by adding either path quantifiers (Section 1.4.1) or language-based temporal operators (regular or context-free languages) following the approach introduced in [21]. We recall the main expressive power and complex-

ity issues related to Extended Temporal Logic ETL [21] (Section 1.4.2). Finally, we show that model-checking for propositional calculus augmented with a simple context-free language recognized by a visibly pushdown automaton (VPA) [22] is highly undecidable (Section 1.4). It is worth observing that Propositional Dynamic Logic with programs [23] (PDL) augmented with visibly pushdown automata has been recently shown decidable [24] and the class of VPA shares very nice features (closure under Boolean operations for instance [22]).

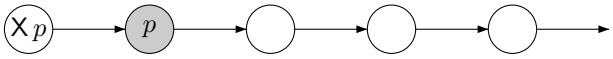
1.2. Temporal logics

1.2.1. *Modalities about executions*

The languages of TL contain precisely modalities having a temporal interpretation. A modality is usually defined as a syntactic object (term) that modifies the relationships between a predicate and a subject. For example, in the sentence "Tomorrow, it will rain", the term "Tomorrow" is a temporal modality. TL makes use of different types of modalities and we recall below some of them interpreted over runs (a.k.a. executions or ω-sequences). The temporal modalities (a.k.a. temporal combinators) allow one to speak about the sequencing of states along an execution, rather than about the states taken individually. The simplest temporal combinators are X ("neXt"), F ("sometimes") and G ("always"). Below, we shall freely use the Boolean operators $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction) and $\rightarrow$ (material implication).

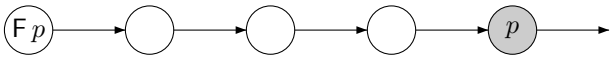
- Whereas φ states a property of the current state, $\mathsf{X}\,\varphi$ states that the next state (X for "neXt") satisfies φ. For example, $\varphi \vee \mathsf{X}\,\varphi$ states that φ is satisfied now or in the next state.

  $\mathsf{X}\,p$: next-time p

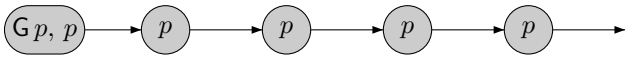
- $\mathsf{F}\,p$ announces that a future state (F for "Future") satisfies φ without specifying which state, and $\mathsf{G}\,\varphi$ that all the future states satisfy φ. These two combinators can be read informally as "φ will hold some day" and "φ will always be". Foundations of the modal approach to temporal logic can be found in [25] in which are introduced the combinators F and G.

  $\mathsf{F}\,p$: sometimes p

  Duality The operator G is the dual of F: whatever the formula φ may be, if φ is always satisfied, then it is not true that $\neg\varphi$ will some day be satisfied, and conversely. Hence $\mathsf{G}\,\varphi$ and $\neg\,\mathsf{F}\,\neg\varphi$ are equivalent.

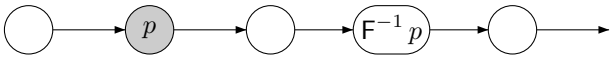
$\mathsf{G}\,p$: always p



By way of example, the expression $\texttt{alert} \rightarrow \mathsf{F}\ \texttt{halt}$ means that if we (currently) are in a state of alert, then we will (later) be in a halt state.

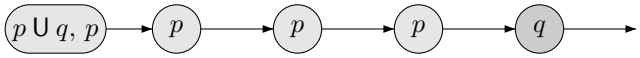
Past-time operators Likewise, the past operators F^{-1} ("sometime in the past") and G^{-1} ("always in the past") are also introduced in [25].

$\mathsf{F}^{-1}\,p$: sometime in the past p



- The U combinator (for "Until") is richer and more complicated than the combinator F. $\varphi_1\,\mathsf{U}\,\varphi_2$ states that φ_1 is true until φ_2 is true. More precisely: φ_2 will be true some day, and φ_1 will hold in the meantime.

$p\,\mathsf{U}\,q$: p until q



The example $\mathsf{G}(\texttt{alert} \rightarrow \mathsf{F}\ \texttt{halt})$ can be refined with the statement that "starting from a state of alert, the alarm remains activated until the halt state is eventually reached":

$$\mathsf{G}(\texttt{alert} \rightarrow (\texttt{alarm}\ \mathsf{U}\ \texttt{halt})).$$

Sometime operator The F combinator is a special case of U: $\mathsf{F}\,\varphi$ and $\texttt{true}\,\mathsf{U}\,\varphi$ are equivalent.

Weak until There exists also a "weak until", denoted W. The statement $\varphi_1\,\mathsf{W}\,\varphi_2$ still expresses "$\varphi_1\,\mathsf{U}\,\varphi_2$", but without the inevitable occurrence of φ_2 and if φ_2 never occurs, then φ_1 remains true forever. So, $\varphi_1\,\mathsf{W}\,\varphi_2$ is equivalent to $\mathsf{G}\,\varphi_1 \vee (\varphi_1\,\mathsf{U}\,\varphi_2)$.

Release operator In the sequel, we shall also use the so-called "release" operator R which is the dual of the until operator U. The formula $\varphi_1\,\mathsf{R}\,\varphi_2$ states that the truth of φ_1 releases the constraint on the satisfaction of φ_2, more precisely, either φ_1 will be true some day and φ_2 must hold between the current state and that day, or φ_2 must be true in all future states.

We provide below other examples of properties that can be expressed thanks to these temporal modalities.

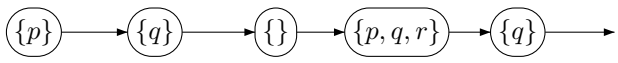
(safety) $\mathsf{G}(\texttt{halt} \rightarrow \mathsf{F}^{-1}\,\texttt{alert})$,
(liveness) $\mathsf{G}(p \rightarrow \mathsf{F}\,q)$,
(total correctness) $(\texttt{init} \wedge p) \rightarrow \mathsf{F}(\texttt{end} \wedge q)$,
(strong fairness) $\mathsf{G}\,\mathsf{F}\,\texttt{enabled} \rightarrow \mathsf{G}\,\mathsf{F}\,\texttt{executed}$.

1.2.2. *Linear-time temporal logic* LTL

As far as we know, linear-time temporal logic LTL in the form presented herein has been first considered in [26] based on the early works [3, 27]. Indeed, the strict until operator in [27] can express the temporal operators from LTL (without past-time operators) and in [3] temporal logics are advocated for the formal verification of programs. However, the version of LTL with explicitly the next-time and until operators first appeared in [26]. Actually, the next-time operator has been introduced in [28] in order to define LTL restricted to the next-time and sometime operators (see also a similar language in [29]). Nowadays, LTL is one of the most used logical formalisms to specify the behaviours of computer systems in view of formal verification. It has been also the basis for numerous specification languages such as PSL [30]. Moreover, it is used as a specification language in tools such as SPIN [31] and SMV [32]. LTL formulae are built from the following abstract grammar:

$$\varphi ::= \overbrace{\bot \mid \top \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi}^{\text{propositional calculus}} \mid \overbrace{\mathsf{X}\,\varphi \mid \mathsf{F}\,\varphi \mid \mathsf{G}\,\varphi \mid \varphi\,\mathsf{U}\,\psi \mid \varphi\,\mathsf{R}\,\psi}^{\text{temporal extension}}$$

where p ranges over a countably infinite set AP of propositional variables. Elements of AP are obtained by abstracting properties, for instance p may mean "$x = 0$". Given a set of temporal operators $\mathcal{O} \subseteq \{\mathsf{X}, \mathsf{F}, \mathsf{G}, \mathsf{U}, \mathsf{R}\}$, we write LTL($\mathcal{O}$) to denote the restriction of LTL to formulae with temporal connectives from $\mathcal{O}$. We write $\mathrm{sub}(\varphi)$ to denote the set of subformulae of the formula φ and $|\varphi|$ to denote the size of the formula φ viewed as a string of characters. LTL models are program runs, i.e., executions viewed as ω-sequences. The reason why the models are infinite objects (instead of finite sequences to encode finite runs) is mainly due to the possibility to specify limit behaviours such as fair behaviours. So, a structure (or model) for LTL is an infinite sequence $u : \mathbb{N} \rightarrow 2^{\mathrm{AP}}$, i.e., an infinite word of $(2^{\mathrm{AP}})^{\omega}$. Here are the five first states of an LTL model:



Given a structure u, a position $i \in \mathbb{N}$ and a formula φ, we define inductively the satisfaction relation $\models$ as follows:

- always $u, i \models \top$ and never $u, i \models \bot$,
- $u, i \models p \stackrel{\text{def}}{\Leftrightarrow} p \in u(i)$, for every $p \in \mathrm{AP}$,
- $u, i \models \neg\varphi \stackrel{\text{def}}{\Leftrightarrow} u, i \not\models \varphi$,
- $u, i \models \varphi_1 \wedge \varphi_2 \stackrel{\text{def}}{\Leftrightarrow} u, i \models \varphi_1$ and $u, i \models \varphi_2$,

- $u, i \models \varphi_1 \vee \varphi_2 \stackrel{\text{def}}{\Leftrightarrow} u, i \models \varphi_1$ or $u, i \models \varphi_2$,
- $u, i \models \mathsf{X}\,\varphi \stackrel{\text{def}}{\Leftrightarrow} u, i+1 \models \varphi$,
- $u, i \models \mathsf{F}\,\varphi \stackrel{\text{def}}{\Leftrightarrow}$ there is $j \geq i$ such that $u, j \models \varphi$,
- $u, i \models \mathsf{G}\,\varphi \stackrel{\text{def}}{\Leftrightarrow}$ for all $j \geq i$, we have $u, j \models \varphi$,
- $u, i \models \varphi_1\,\mathsf{U}\,\varphi_2 \stackrel{\text{def}}{\Leftrightarrow}$ there is $j \geq i$ such that $u, j \models \varphi_2$ and $u, k \models \varphi_1$ for all $i \leq k < j$,
- $u, i \models \varphi_1\,\mathsf{R}\,\varphi_2 \stackrel{\text{def}}{\Leftrightarrow} u, j \models \varphi_2$ for all $j \geq i$, or there is $j \geq i$ such that $u, j \models \varphi_1$ and $u, k \models \varphi_2$ for all $i \leq k \leq j$.

We say that two formulae φ and ψ are *equivalent* whenever for all models u and positions i, we have $u, i \models \varphi$ if and only if $u, i \models \psi$. In that case, we write $\varphi \equiv \psi$. Roughly speaking, φ and ψ state equivalent properties over the class of ω-sequences indexed by propositional valuations. For instance, $\mathsf{F}\,\varphi$ is equivalent to $\top\,\mathsf{U}\,\varphi$ and $\mathsf{G}\,\varphi$ is equivalent to $\bot\,\mathsf{R}\,\varphi$. Consequently, it is clear that our set of connectives is not minimal in terms of expressive power but it provides handy notations. Moreover, we shall use the following abbreviations: $\varphi_1 \rightarrow \varphi_2$ for $\neg\varphi_1 \vee \varphi_2$ and $\mathsf{F}^{\infty}\,\varphi$ for $\mathsf{G}\,\mathsf{F}\,\varphi$ ("φ holds infinitely often"). Finally, one can check that G is the dual of F (since $\mathsf{G}\,p \equiv \neg\,\mathsf{F}\,\neg p$) and R is the dual of U (since $\varphi_1\,\mathsf{R}\,\varphi_2 \equiv \neg(\neg\varphi_1\,\mathsf{U}\,\neg\varphi_2)$).

We write $u \models \varphi$ instead of $u, 0 \models \varphi$. The standard automata-based approach for LTL, see e.g. [9], considers the models for a formula φ as a language $\mathcal{L}(\varphi)$ over the alphabet $2^{\mathrm{AP}(\varphi)}$ where $\mathrm{AP}(\varphi)$ denotes the set of propositional variables occurring in φ (these are the only relevant ones for the satisfaction of φ):

$$\mathcal{L}(\varphi) = \{u \in (2^{\mathrm{AP}(\varphi)})^{\omega} \mid u \models \varphi\}.$$

We say that φ is satisfiable if $\mathcal{L}(\varphi)$ is non-empty. Similarly, φ is valid if $\mathcal{L}(\neg\varphi)$ is empty. The satisfiability problem for LTL, denoted by SAT(LTL), is defined as follows:

input: an LTL formula φ,
output: 1 if $u \models \varphi$ for some infinite word $u \in (2^{\mathrm{AP}(\varphi)})^{\omega}$; 0 otherwise.

The validity problem VAL(LTL) is defined similarly. In order to be precise, it is worth observing that herein we consider *initial* satisfiability since the formula φ holds at the position 0. Since LTL (and the other temporal logics considered in this chapter) does not deal with past-time operators, *initial* satisfiability is equivalent to satisfiability (satisfaction at some position, not necessarily 0).

Let us now consider the model-checking problem. A Kripke structure $\mathcal{M} = \langle W, R, \lambda\rangle$ is a triple such that

- W is a non-empty set of states,
- R is a binary relation on W (accessibility relation, one-step relation),
- λ is a labeling $\lambda : W \rightarrow 2^{\mathrm{AP}}$.

$\mathcal{M}$ is simply a directed graph for which each node is labeled by a propositional interpretation (labeled transition system). A path in $\mathcal{M}$ is a sequence

$\sigma = s_0 s_1 s_2 \cdots$ (finite or infinite) such that $(s_i, s_{i+1}) \in R$ for every $i \geq 0$. We write $\mathrm{Paths}(\mathcal{M}, s_0)$ to denote the set of infinite paths of $\mathcal{M}$ starting at state s_0. We also write $\lambda\mathrm{Paths}(\mathcal{M}, s_0)$ to denote the set of *labels* of infinite paths starting at s_0: $\lambda\mathrm{Paths}(\mathcal{M}, s_0) = \{\lambda(s_0)\lambda(s_1)\lambda(s_2)\cdots \mid s_0 s_1 s_2 \cdots \in \mathrm{Paths}(\mathcal{M}, s_0)\}$.

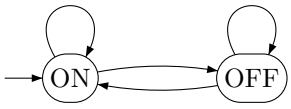
The (*universal*) model-checking problem for LTL, denoted by $\mathrm{MC}^{\forall}(\mathrm{LTL})$, is defined as follows:

input: an LTL formula φ, a finite and total[†] Kripke structure $\mathcal{M}$ and $s_0 \in W$,
output: 1 if $u \models \varphi$ for all $u \in \lambda\mathrm{Paths}(\mathcal{M}, s_0)$ (written $\mathcal{M}, s_0 \models_\forall \varphi$); 0 otherwise.

Without any loss of generality, in the above statement we can assume that the codomain of the labeling λ is restricted to $\mathrm{AP}(\varphi)$. The size of $\langle W, R, \lambda\rangle$ is defined by $\mathrm{card}(W) + \mathrm{card}(R) + \Sigma_{w\in W}\mathrm{card}(\lambda(w))$. It is easy to check that $\mathcal{M}, s_0 \models_\forall \varphi$ if and only if $\lambda\mathrm{Paths}(\mathcal{M}, s_0) \cap \mathcal{L}(\neg\varphi) = \emptyset$.

There is a dual definition, called *existential* model checking and denoted by $\mathrm{MC}^{\exists}(\mathrm{LTL})$, where an existential quantification on paths is considered. We write $\mathcal{M}, s_0 \models_\exists \varphi$ if $u \models \varphi$ for some $u \in \lambda\mathrm{Paths}(\mathcal{M}, s_0)$. Similarly, $\mathcal{M}, s_0 \models_\exists \varphi$ if and only if $\lambda\mathrm{Paths}(\mathcal{M}, s_0) \cap \mathcal{L}(\varphi) \neq \emptyset$.

We present below a Kripke structure in which ON and OFF are propositional variables and we identify them with states where they hold respectively.



We leave to the reader to check that the properties below hold:

- $\mathcal{M}, \mathrm{ON} \models_\exists \mathsf{F}^\infty \mathrm{ON} \wedge \mathsf{F}^\infty \mathrm{OFF}$,
- $\mathcal{M}, \mathrm{ON} \models_\exists \neg\, \mathsf{F}^\infty \mathrm{OFF}$,
- $\mathcal{M}, \mathrm{ON} \models_\exists \mathsf{G}(\mathrm{ON} \rightarrow \mathsf{X}\mathsf{X}\ \mathrm{OFF})$.

1.2.3. *Branching-time temporal logic* CTL*

The language introduced so far can only state properties along one execution. It is also often desirable to express the branching aspect of the behavior: many futures are possible starting from a given state. For instance, consider the property φ which is informally defined as: "whenever we are in a state where p holds, it is possible to reach a state where q holds". This natural property cannot be expressed in LTL. Indeed, with the models $\mathcal{M}_1$ and $\mathcal{M}_2$ of Figure 1.1, we have $\lambda\mathrm{Paths}(\mathcal{M}_1) = \lambda\mathrm{Paths}(\mathcal{M}_2)$. Hence $\mathcal{M}_1$ and $\mathcal{M}_2$ satisfy the same LTL formulae. But indeed, $\mathcal{M}_1$ satisfies φ whereas $\mathcal{M}_2$ does not.

The logic CTL* introduces special purpose quantifiers, A (compare with $\forall$ in first-order logic) and E (compare with $\exists$ in first-order logic), which allow to quantify over the set of executions. These are called *path quantifiers*. The expression $\mathsf{A}\,\varphi$

[†] $\forall x \in W, \exists y \in W,\ \langle x, y\rangle \in R$.

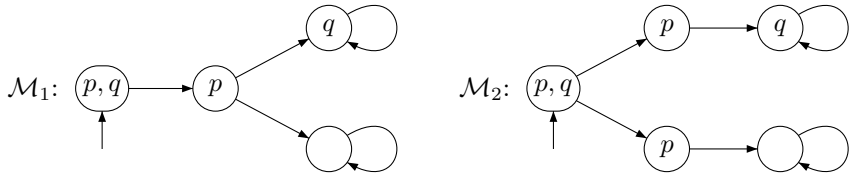
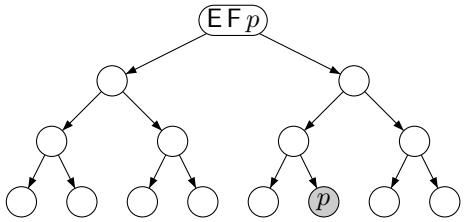
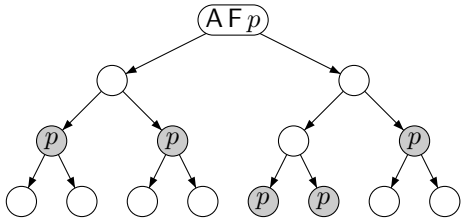


Fig. 1.1. Two models undistinguishable for LTL.

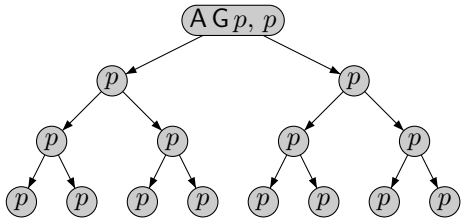
states that all executions out of the current state satisfy property φ. Dually, $\mathsf{E}\,\varphi$ states that from the current state, there exists an execution satisfying φ. Our "natural" property above can be written $\mathsf{A}\,\mathsf{G}(p \to \mathsf{E}\,\mathsf{F}\,q)$. It is worth observing that A and E quantify over paths whereas G and F quantify over positions along a path. The expression $\mathsf{E}\,\mathsf{F}\,\varphi$ states that it is possible (by following a suitable execution) to have φ some day, which is illustrated below.



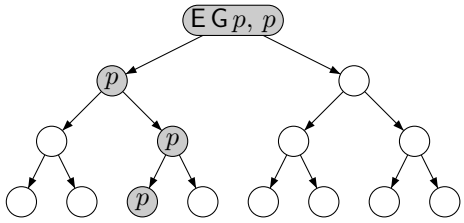
The expression $\mathsf{A}\,\mathsf{F}\,\varphi$ states that we will necessarily have φ some day, regardless of the chosen execution (see below).



The expression $\mathsf{A}\,\mathsf{G}\,\varphi$ states that φ holds in all future states including now (see below).



The expression $\mathsf{E}\,\mathsf{G}\,\varphi$ states that there is an execution on which φ always holds (see below).



"Branching-time logics" refers to logics that have the ability to freely quantify over paths. Standard examples of branching-time temporal logics include Computation Tree Logic CTL [33], CTL* [34] and the modal μ-calculus. We define below the logic CTL* (that is more expressive than both LTL and CTL) for which the model-checking problem can be solved easily by using a subroutine solving the model-checking problem for LTL. Hence, even though the object of this chapter is not especially dedicated to branching-time logics, we explain how CTL* model-checking can be dealt with. CTL* formulae are built from the following abstract grammar:

$$\varphi \ ::= \ p \ \mid \ \neg\varphi \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \mathsf{E}\,\varphi \ \mid \ \mathsf{A}\,\varphi \ \mid \ \mathsf{X}\,\varphi \ \mid \ \varphi_1 \,\mathsf{U}\, \varphi_2$$

where p ranges over AP. CTL* models are total Kripke models. Let $\sigma = s_0 s_1 \ldots$ be an infinite path in $\mathcal{M}$, $i \geq 0$ and φ be a formula. The satisfaction relation $\sigma, i \models \varphi$ is defined inductively as follows (we omit the clauses for Boolean connectives):

- $\sigma, i \models p \stackrel{\text{def}}{\Leftrightarrow} p \in \lambda(s_i)$,
- $\sigma, i \models \mathsf{X}\,\varphi \stackrel{\text{def}}{\Leftrightarrow} \sigma, i+1 \models \varphi$,
- $\sigma, i \models \varphi_1 \,\mathsf{U}\, \varphi_2 \stackrel{\text{def}}{\Leftrightarrow}$ there is $j \geq i$ such that $\sigma, j \models \varphi_2$ and $\sigma, k \models \varphi_1$ for all $i \leq k < j$,
- $\sigma, i \models \mathsf{E}\,\varphi \stackrel{\text{def}}{\Leftrightarrow}$ there is an infinite path $\sigma' = s'_0 s'_1 \ldots$ such that $s'_0 = s_i$ and $\sigma', 0 \models \varphi$,
- $\sigma, i \models \mathsf{A}\,\varphi \stackrel{\text{def}}{\Leftrightarrow}$ for every infinite path $\sigma' = s'_0 s'_1 \ldots$ such that $s'_0 = s_i$, we have $\sigma', 0 \models \varphi$.

The model-checking problem for CTL*, denoted by MC$^\forall$(CTL*), is defined as follows:

input: a CTL* formula, a finite and total Kripke model $\mathcal{M} = \langle W, R, \lambda \rangle$ and $s \in W$;
output: 1 if $\sigma, 0 \models \varphi$ for all infinite paths $\sigma \in \mathrm{Paths}(\mathcal{M}, s)$ starting from s; 0 otherwise.

1.2.4. *Complexity issues*

Let us recall a few complexity results.

Theorem 1.1. *[34–37] The following problems are* PSPACE*-complete.*

(i) SAT(LTL)*,* VAL(LTL)*,* MC$^\exists$(LTL) *and* MC$^\forall$(LTL)*.*

(ii) $\mathrm{SAT}(\mathrm{LTL}(\mathsf{X},\mathsf{F}))$, $\mathrm{VAL}(\mathrm{LTL}(\mathsf{X},\mathsf{F}))$, $\mathrm{MC}^{\exists}(\mathrm{LTL}(\mathsf{X},\mathsf{F}))$ *and* $\mathrm{MC}^{\forall}(\mathrm{LTL}(\mathsf{X},\mathsf{F}))$.
(iii) $\mathrm{SAT}(\mathrm{LTL}(\mathsf{U}))$, $\mathrm{VAL}(\mathrm{LTL}(\mathsf{U}))$, $\mathrm{MC}^{\exists}(\mathrm{LTL}(\mathsf{U}))$ *and* $\mathrm{MC}^{\forall}(\mathrm{LTL}(\mathsf{U}))$.
(iv) The restriction of the above problems to a unique propositional variable.
(v) $\mathrm{MC}(\mathrm{CTL}^*)$.

On the other hand, the problems $\mathrm{SAT}(\mathrm{LTL}(\mathsf{F}))$, $\mathrm{MC}^{\exists}(\mathrm{LTL}(\mathsf{F}))$ *are* NP*-complete and* $\mathrm{VAL}(\mathrm{LTL}(\mathsf{F}))$, $\mathrm{MC}^{\forall}(\mathrm{LTL}(\mathsf{F}))$ *are* coNP*-complete.*

The treatment in Section 1.3 will establish that SAT(LTL), VAL(LTL), $\mathrm{MC}^{\forall}(\mathrm{LTL})$ and $\mathrm{MC}^{\exists}(\mathrm{LTL})$ are in PSPACE.

The Computation Tree Logic CTL [33] is a strict fragment of CTL^* for which model-checking can be solved in polynomial-time. We briefly recall that CTL formulae are defined by the grammar below:

$$\varphi \ ::= \ p \ \mid \ \neg\varphi \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \mathsf{E}\,\varphi_1\,\mathsf{U}\,\varphi_2 \ \mid \ \mathsf{A}\,\varphi_1\,\mathsf{U}\,\varphi_2 \ \mid \ \mathsf{EX}\,\varphi \ \mid \ \mathsf{AX}\,\varphi$$

CTL model-checking is PTIME-complete and the complexity function is bilinear in the size of the formula and in the size of the Kripke structure [38] (see also the survey paper [37]) whereas CTL satisfiability is EXPTIME-complete [33]. By contrast, the satisfiability problem for CTL^* is much more complex: 2EXPTIME-complete (upper bound from [39] and lower bound from [40]).

1.3. From LTL formulae to Büchi automata

In this section, we explain how to translate an LTL formula φ into an automaton $\mathcal{A}_\varphi$ such that the language recognized by $\mathcal{A}_\varphi$ is precisely $\mathcal{L}(\varphi)$. However, in order to be of practical use, the translation process can be divided in four stages (at least):

(1) preprocessing the LTL formula using simple logical equivalences,
(2) translation of φ into a generalized Büchi automaton $\mathcal{A}_\varphi$ (the core of the construction),
(3) simplification and optimization of $\mathcal{A}_\varphi$,
(4) translation of $\mathcal{A}_\varphi$ into a Büchi automaton.

Indeed, it is legitimate to aim at building Büchi automata as simple as possible, even though we know that in the worst case the translation has an exponential blow-up. This section is mainly dedicated to step (2) with the construction of simple automata. Considerations about steps (1) and (3) can be found in Sections 1.3.3 and 1.3.6, respectively.

1.3.1. *Automata-based approach*

The construction of $\mathcal{A}_\varphi$ from the formula φ remains the core for the decision procedures of the satisfiability problem and the model checking problem for LTL specifications. Indeed, the (initial) satisfiability problem amounts to checking the Büchi automaton $\mathcal{A}_\varphi$ for emptiness. To solve the model-checking problem, one constructs

first the product $\mathcal{B} = \mathcal{M} \times \mathcal{A}_{\neg\varphi}$ of the model $\mathcal{M}$ with the automaton $\mathcal{A}_{\neg\varphi}$ so that successful runs of $\mathcal{B}$ correspond to infinite runs of $\mathcal{M}$ satisfying the formula $\neg\varphi$. Therefore, $\mathcal{L}(\mathcal{B}) = \emptyset$ if and only if $\mathcal{M} \models_\forall \varphi$ and the model-checking problem is again reduced to the emptiness problem for a Büchi automaton.

Note that checking nonemptiness of a Büchi automaton can be done efficiently (NLogSpace or linear time, see e.g. [41, Theorem 12]) since it reduces to several reachability questions in the underlying graph of the automaton: we have to find a reachable accepting state with a loop around it. Since both the satisfiability problem and the model-checking problem for LTL specifications are PSpace-complete, we cannot avoid an exponential blow-up in the worst case when constructing a Büchi automaton $\mathcal{A}_\varphi$ associated with an LTL formula φ. Fortunately, in most practical cases, we can construct a *small* Büchi automaton $\mathcal{A}_\varphi$.

It is therefore very important to have *good* constructions for the Büchi automaton $\mathcal{A}_\varphi$ even though there are several interpretations of *good*. It is indeed important to obtain a *small* automaton and several techniques have been developed to reduce the size of the resulting automaton [42]. On the other hand, it is also important to have a quick construction. Some constructions, such as the tableau construction [15], may take an exponential time even if the resulting *reduced* automaton is small. The problem with the most efficient constructions [17] is that they are involved, technical and based on more elaborate structures such as alternating automata. Herein we are interested in a *good* translation from a pedagogical point of view. Our construction is a middle term between tableau constructions and more elaborate constructions based on alternating automata. As a result, it will be efficient, it will produce small automata, and it will be possible to translate non trivial LTL formulae by hand. However, we admit that it is neither the most efficient nor the one that produces the smallest automata.

1.3.2. *Büchi automata in a nutshell*

We first recall the definition of Büchi automata (BA) and some useful generalizations; a self-contained introduction to the theory of finite-state automata for infinite words can be found in Chapter **??**. A BA is a tuple $\mathcal{A} = (Q, \Sigma, I, T, F)$ where Q is a finite set of states, Σ is the alphabet, $I \subseteq Q$ is the set of initial states, $T \subseteq Q \times \Sigma \times Q$ is the set of transitions, and $F \subseteq Q$ is the set of accepting (repeated, final) states.

A *run* of $\mathcal{A}$ is a sequence $\rho = s_0, a_0, s_1, a_1, s_2, \ldots$ such that $(s_i, a_i, s_{i+1}) \in T$ is a transition for all $i \geq 0$. The run ρ is *successful* if $s_0 \in I$ is initial and some state of F is repeated infinitely often in ρ: $\inf(\rho) \cap F \neq \emptyset$ where we let $\inf(\rho) = \{s \in Q \mid \forall\, i,\ \exists\, j > i,\ s = s_j\}$. The label of ρ is the word $u = a_0 a_1 \cdots \in \Sigma^\omega$. The automaton $\mathcal{A}$ accepts the language $\mathcal{L}(\mathcal{A})$ of words $u \in \Sigma^\omega$ such that there exists a successful run of $\mathcal{A}$ on the word u, i.e., with label u. For instance, the automaton in Figure 1.2 accepts those words over $\{a, b\}$ having infinitely many a's (the initial states are marked with an incoming arrow and the repeated states are doubly circled).

When dealing with models for an LTL formula φ, the words are over the alphabet

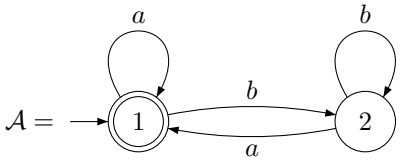


Fig. 1.2. $\mathcal{L}(\mathcal{A}) = \{u \in \{a,b\}^\omega \mid |u|_a = \omega\}$

$\Sigma = 2^{\mathrm{AP}(\varphi)}$ where $\mathrm{AP}(\varphi)$ is the set of propositional variables occurring in φ. A letter $a \in \Sigma$ is read as a propositional valuation for which exactly the propositional variables in a hold. We take advantage of this natural interpretation for defining sets of letters: given a propositional formula ψ, we let $\Sigma_\psi = \{a \in \Sigma \mid a \models \psi\}$ for which "$\models$" refers to the satisfaction relation from propositional calculus. For instance, we have $\Sigma_p = \{a \in \Sigma \mid p \in a\}$, $\Sigma_{\neg p} = \Sigma \setminus \Sigma_p$, $\Sigma_{p \wedge q} = \Sigma_p \cap \Sigma_q$, $\Sigma_{p \vee q} = \Sigma_p \cup \Sigma_q$ and $\Sigma_{p \wedge \neg q} = \Sigma_p \setminus \Sigma_q$. In general, a transition between two states s, s' will be enabled for all letters satisfying some propositional formula ψ. We use $s \xrightarrow{\Sigma_\psi} s'$ as a concise representation of the set of transitions $\{s \xrightarrow{a} s' \mid a \in \Sigma_\psi\}$.

Several examples of Büchi automata corresponding to LTL formulae are given in Figure 1.3. In these automata, transitions are labeled with subsets of Σ meaning that all letters in the subset are allowed for the transition. In some cases, the automaton associated with a formula is deterministic, that is for all $s \in Q$ and $a \in \Sigma$, $\{s' \mid \langle s, a, s' \rangle \in T\}$ has at most one state. Although, determinism is a very desirable property, it is not always possible. For instance, the automaton for $\mathsf{G}\,\mathsf{F}\,p$ is deterministic whereas the automaton for its negation $\neg\,\mathsf{G}\,\mathsf{F}\,p \equiv \mathsf{F}\,\mathsf{G}\,\neg p$ must be nondeterministic. This is an easy example showing that deterministic BA are not closed under complement.

By contrast, Büchi automata are closed under union, intersection and complement, which corresponds to the Boolean operations on formulae. It is also easy to construct an automaton for $\mathsf{X}\,\varphi$ from an automaton for φ, see for instance the automaton for $\mathsf{X}\,\mathsf{X}\,p$ in Figure 1.3. Finally, one can construct an automaton for $\varphi\,\mathsf{U}\,\psi$ from automata for φ and ψ. Hence, we have a modular construction of $\mathcal{A}_\varphi$ for any LTL formula φ. But both negation and until yield an exponential blowup. Hence this modular construction is non-elementary and useless in practice, see also [43, 44].

Now we introduce a generalization of the acceptance condition of Büchi automata. First, it will fall on transitions instead of states, as considered also in [18]. Second it will allow conjunctions of classical Büchi conditions. Formally, a generalized Büchi automaton (GBA) is a tuple $\mathcal{A} = (Q, \Sigma, I, T, T_1, \ldots, T_n)$ where Q, Σ, I, T are as above and the acceptance condition which deals with transitions is given by the sets $T_i \subseteq T$ for $1 \leq i \leq n$. For a run ρ of $\mathcal{A}$, we denote by $\inf_T(\rho)$ the set of transitions that occur infinitely often in ρ and the run is *successful* if $\inf_T(\rho) \cap T_i \neq \emptyset$ for each $1 \leq i \leq n$. Often, we simply write $\inf(\rho)$ instead of $\inf_T(\rho)$. For instance, a GBA is given in Figure 1.4 where we require both that transitions with short dashes and transitions with long dashes are repeated infinitely often.

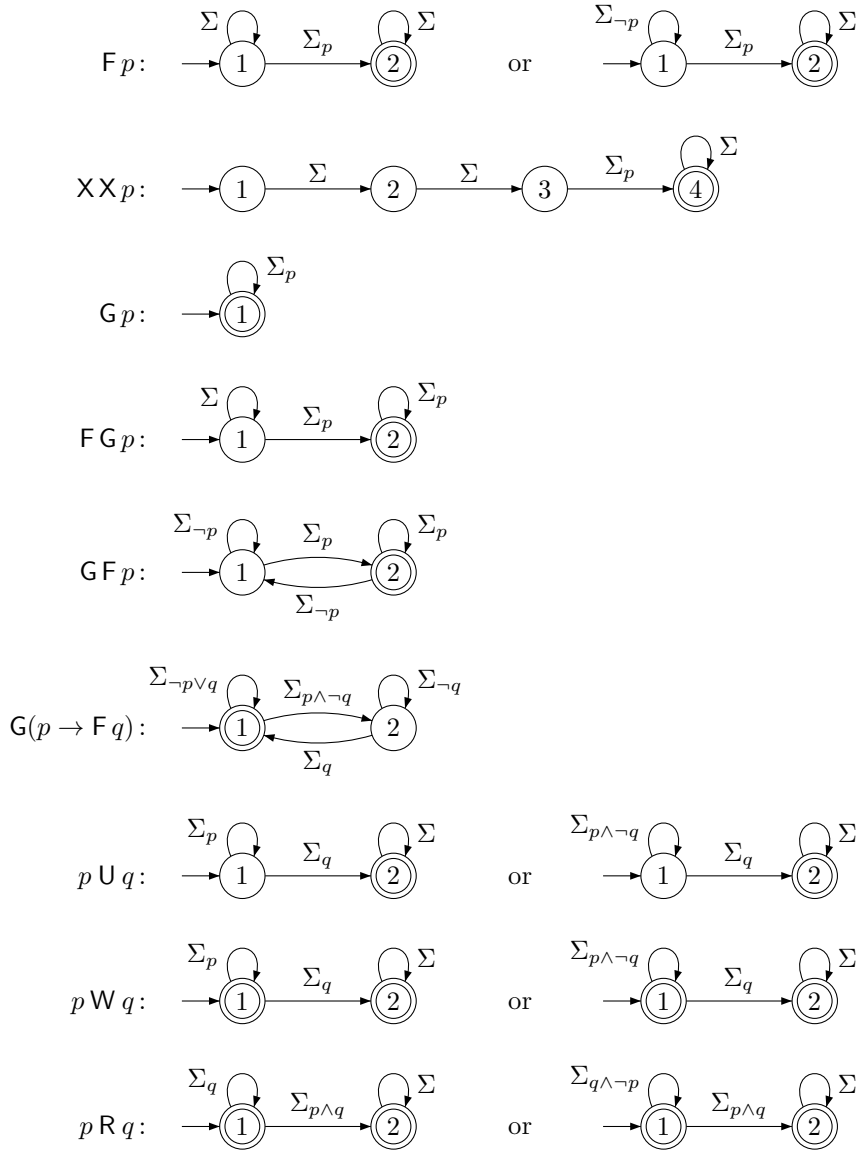


Fig. 1.3. Büchi automata for some LTL formulae

Each GBA $\mathcal{A}$ can be easily translated into a classical BA (preserving the language of accepted ω-words). Indeed, it is sufficient to perform a synchronized product $\mathcal{A} \otimes \mathcal{B}$ with the automaton $\mathcal{B}$ in Figure 1.5 where the accepting Büchi states in the product are those containing the state n in the second component. Synchro-

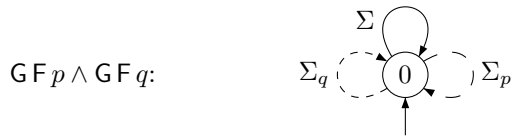


Fig. 1.4. Generalized Büchi automaton

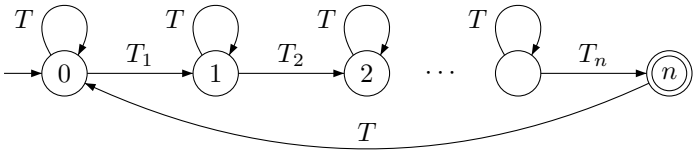


Fig. 1.5. Transforming a GBA into a classical BA

nization is performed thanks to the rule below:

$$\frac{t = s_1 \xrightarrow{a} s_1' \in \mathcal{A} \qquad s_2 \xrightarrow{t} s_2' \in \mathcal{B}}{\langle s_1, s_2\rangle \xrightarrow{a} \langle s_1', s_2'\rangle \in \mathcal{A} \otimes \mathcal{B}}$$

Note that the intended construction can be performed in logarithmic space since typically in order to build the automaton in Figure 1.5 one needs a counter of size $\mathcal{O}(log(n))$ and in order to address some part of the GBA $\mathcal{A}$ (in order to build the product) one needs a register of size $\mathcal{O}(log(|\mathcal{A}|))$.

1.3.3. *Preprocessing the* LTL *formula*

We have now all the background on Büchi automata that are useful for our construction of a GBA associated with an LTL formula. The first step is to put the formula in *negative normal form*, i.e., to propagate the negation connectives inwards. This can be done while preserving logical equivalence since all the connectives have a dual connective in LTL (X is self-dual). The equivalences below can be read as rewriting rules from left to right:

$$\begin{array}{cc} \neg(\varphi \vee \psi) \equiv (\neg\varphi) \wedge (\neg\psi) & \neg(\varphi \wedge \psi) \equiv (\neg\varphi) \vee (\neg\psi) \\ \neg(\varphi \,\mathsf{U}\, \psi) \equiv (\neg\varphi)\,\mathsf{R}\,(\neg\psi) & \neg(\varphi \,\mathsf{R}\, \psi) \equiv (\neg\varphi)\,\mathsf{U}\,(\neg\psi) \\ \neg\,\mathsf{X}\,\varphi \equiv \mathsf{X}\,\neg\varphi & \neg\neg\varphi \equiv \varphi \end{array}$$

Formally, an LTL formula is in *negative normal form* (NNF) if it follows the syntax given by

$$\varphi ::= \top \mid \bot \mid p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\varphi \mid \varphi\,\mathsf{R}\,\varphi$$

where p ranges over atomic propositions in AP.

In the following, a *temporal* formula is defined as either a literal (i.e., a propositional variable or its negation) or a formula in NNF with outermost connective

among $\{\mathsf{X}, \mathsf{U}, \mathsf{R}\}$. Therefore, any LTL formula in NNF is a positive Boolean combination of temporal formulae. Note that, translating an arbitrary LTL formula in NNF does not increase the number of temporal subformulae. This is important since the size of the GBA $\mathcal{A}_\varphi$ that we will construct depends on the number of temporal subformulae of φ. Therefore, before starting the construction, it is useful to rewrite the formula in order to reduce the number of temporal subformulae. Several rewriting rules are presented in [45, 46] and we only give below some examples which again should be applied from left to right.

$$(\mathsf{X}\,\varphi) \wedge (\mathsf{X}\,\psi) \equiv \mathsf{X}(\varphi \wedge \psi) \qquad (\mathsf{X}\,\varphi)\,\mathsf{U}\,(\mathsf{X}\,\psi) \equiv \mathsf{X}(\varphi\,\mathsf{U}\,\psi)$$
$$(\varphi\,\mathsf{R}\,\psi_1) \wedge (\varphi\,\mathsf{R}\,\psi_2) \equiv \varphi\,\mathsf{R}\,(\psi_1 \wedge \psi_2) \qquad (\varphi_1\,\mathsf{R}\,\psi) \vee (\varphi_2\,\mathsf{R}\,\psi) \equiv (\varphi_1 \vee \varphi_2)\,\mathsf{R}\,\psi$$
$$(\mathsf{G}\,\varphi) \wedge (\mathsf{G}\,\psi) \equiv \mathsf{G}(\varphi \wedge \psi) \qquad \mathsf{G}\,\mathsf{F}\,\varphi \vee \mathsf{G}\,\mathsf{F}\,\psi \equiv \mathsf{G}\,\mathsf{F}(\varphi \vee \psi)$$

It is worth noting that the above simplification rules are useful in practice. By contrast, writing a formula in NNF remains a step that mainly eases the presentation of the forthcoming construction. Indeed, propagating the negation connectives inwards can be performed symbolically by storing subformulae of the initial formula augmented with polarities in $\{0, 1\}$.

1.3.4. *Building simple automata*

We start now the description of the core of our construction. A state Z of an automaton $\mathcal{A}_\varphi$ will be a subset of $\mathrm{sub}(\varphi)$, the set of subformulae of φ. We say that a set Z of formulae is *consistent* if it does not contain $\bot$ or a pair $\{\psi, \neg\psi\}$ for some formula ψ (since our formulae are in NNF, ψ could only be a propositional variable). We often need the conjunction of formulae in Z which will be written $\bigwedge Z = \bigwedge_{\psi \in Z} \psi$. Note that $\bigwedge \emptyset = \top$. The formulae in Z are viewed as *obligations*, i.e., if a run ρ on a word u starts from Z and satisfies the acceptance condition then $u \models \bigwedge Z$, i.e., $u \models \psi$ for all $\psi \in Z$. More precisely, if we denote by $\mathcal{A}_\varphi^Z$ the automaton $\mathcal{A}_\varphi$ where Z is the unique initial state, then our construction will guarantee that

$$\mathcal{L}(\mathcal{A}_\varphi^Z) = \{u \in \Sigma^\omega \mid u \models \bigwedge Z\}$$

Therefore, the unique initial state of $\mathcal{A}_\varphi$ will be the singleton set $\{\varphi\}$.

We say that a set Z of LTL formulae in NNF is *reduced* if all formulae in Z are either literals or formulae with outermost connective X. Given a *consistent* and *reduced* set Z, we write $\mathrm{next}(Z)$ to denote the set $\{\psi \mid \mathsf{X}\,\psi \in Z\}$ and Σ_Z to denote the set of letters which satisfy all literals in Z:

$$\Sigma_Z = \bigcap_{p \in Z} \Sigma_p \;\cap \bigcap_{\neg p \in Z} \Sigma_{\neg p}$$

Equivalently, Σ_Z is the set of letters $a \in \Sigma$ such that for every $p \in \mathrm{AP}(\varphi)$, $p \in Z$ implies $p \in a$ and $\neg p \in Z$ implies $p \notin a$. From a consistent and reduced set Z, the automaton is ready to perform any transition of the form $Z \xrightarrow{a} \mathrm{next}(Z)$ with $a \in \Sigma_Z$.

Table 1.1. Reduction rules.

If $\psi = \psi_1 \wedge \psi_2$:	$Y \xrightarrow{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_1, \psi_2\}$
If $\psi = \psi_1 \vee \psi_2$:	$Y \xrightarrow{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_1\}$ $Y \xrightarrow{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2\}$
If $\psi = \psi_1 \,\mathsf{R}\, \psi_2$:	$Y \xrightarrow{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_1, \psi_2\}$ $Y \xrightarrow{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2, \mathsf{X}\,\psi\}$
If $\psi = \mathsf{G}\,\psi_2$:	$Y \xrightarrow{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2, \mathsf{X}\,\psi\}$
If $\psi = \psi_1 \,\mathsf{U}\, \psi_2$:	$Y \xrightarrow{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2\}$ $Y \xrightarrow[!\psi]{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_1, \mathsf{X}\,\psi\}$
If $\psi = \mathsf{F}\,\psi_2$:	$Y \xrightarrow{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2\}$ $Y \xrightarrow[!\psi]{\varepsilon} Y \setminus \{\psi\} \cup \{\mathsf{X}\,\psi\}$

For instance, for every $a \in \Sigma$, $\emptyset \xrightarrow{a} \emptyset$ is a transition which can be interpreted as: when no obligations have to be satisfied, any letter can be read and there are still no obligations. Note that $\Sigma_Z \neq \emptyset$ since Z is consistent, but $\mathrm{next}(Z)$ is not reduced in general. We will use ε-transitions to reduce arbitrary sets of formulae in reduced sets so that the semantics of the automaton is preserved. These transitions are handy but they will not belong to the final GBA $\mathcal{A}_\varphi$. So let Y be a set of formulae which is not reduced and choose some $\psi \in Y$ *maximal* among the non-reduced formulae in Y (here maximal is for the *subformula* ordering). Depending on the form of ψ, the ε-transitions allowing to reduce ψ are presented in Table 1.1. The rules for G and F can indeed be derived from those for R and U, they are included for convenience. Indeed, we only introduce transitions between *consistent* sets. While $\xrightarrow{\varepsilon}$ denotes the one-step reduction relation, as usual, we write $\xrightarrow[*]{\varepsilon}$ to denote the reflexive and transitive closure of $\xrightarrow{\varepsilon}$.

When ψ is a conjunction or a G formula then we introduce only one ε-transition $Y \xrightarrow{\varepsilon} Y_1$ and $\bigwedge Y \equiv \bigwedge Y_1$. In the other cases, we introduce two ε-transitions $Y \xrightarrow{\varepsilon} Y_1$ and $Y \xrightarrow{\varepsilon} Y_2$ and $\bigwedge Y \equiv \bigwedge Y_1 \vee \bigwedge Y_2$.

We introduce these ε-transitions iteratively until all states have been reduced. The construction terminates since each step removes a *maximal* non-reduced formula and introduces only strictly smaller non-reduced formulae (note that $\mathsf{X}\,\alpha$ is not smaller than α but is reduced).

Finally, note the mark $!\psi$ on the second transitions for U and F. It denotes the fact that the *eventuality* ψ_2 has been postponed. The marked transitions will be used to define the acceptance condition of the GBA in such a way that all eventualities are satisfied along an accepting run.

In Figure 1.6 we show the ε-transitions that are introduced when we start with a singleton set $\{\varphi\}$ with $\varphi = \mathsf{G}(p \rightarrow \mathsf{F}\,q) \equiv \bot\,\mathsf{R}\,(\neg p \vee (\top\,\mathsf{U}\,q))$. Note again the mark $!\mathsf{F}\,q$ on the last transition.

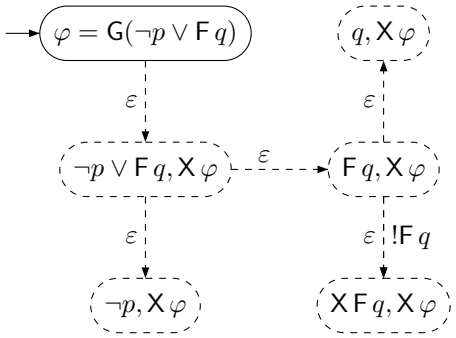


Fig. 1.6. Reduction of a state to reduced states

An *until formula* α of φ (in NNF) is a subformula of φ with outermost connective either U or F. The set of until formulae of φ is denoted by $\mathsf{U}(\varphi)$. For each subset Y of formulae in NNF we define

$$\mathrm{Red}(Y) = \{Z \text{ consistent and reduced} \mid Y \xrightarrow[*]{\varepsilon} Z\}$$

and for each $\alpha \in \mathsf{U}(\varphi)$ we also define

$$\mathrm{Red}_\alpha(Y) = \{Z \text{ consistent and reduced} \mid Y \xrightarrow[*]{\varepsilon} Z \text{ without using an edge marked with } !\alpha\}$$

Thanks to the nice properties of the reduction rules, we obtain the equivalence

$$\bigwedge Y \equiv \bigvee_{Z \in \mathrm{Red}(Y)} \bigwedge Z$$

Consequently, by using the reductions from Figure 1.6 we obtain

$$\mathrm{Red}(\{\varphi\}) = \{\{\neg p, \mathsf{X}\,\varphi\}, \{q, \mathsf{X}\,\varphi\}, \{\mathsf{X}\,\mathsf{F}\, q, \mathsf{X}\,\varphi\}\}$$
$$\mathrm{Red}_{\mathsf{F}\, q}(\{\varphi\}) = \{\{\neg p, \mathsf{X}\,\varphi\}, \{q, \mathsf{X}\,\varphi\}\}$$

Observe that in $\mathrm{Red}_{\mathsf{F}\, q}(\{\varphi\})$, the subscript $\mathsf{F}\, q$ refers to an *absence* of ε-transitions marked by $!\mathsf{F}\, q$ along the reduction path. This is the case when we do not have the obligation $\mathsf{F}\, q$ or if the eventuality $\mathsf{F}\, q$ is satisfied now by imposing the obligation q. By contrast, an ε-transition $\{\mathsf{F}\, q, \mathsf{X}\,\varphi\} \xrightarrow[!\mathsf{F}\, q]{\varepsilon} \{\mathsf{X}\,\mathsf{F}\, q, \mathsf{X}\,\varphi\}$ with mark $!\mathsf{F}\, q$ indicates that the eventuality $\mathsf{F}\, q$ is *not* satisfied now. We hope this is not too confusing.

We give now the formal definition of $\mathcal{A}_\varphi = (Q, \Sigma, I, T, (T_\alpha)_{\alpha \in \mathsf{U}(\varphi)})$. The set of states is $Q = 2^{\mathrm{sub}(\varphi)}$ and the initial state is the singleton $I = \{\varphi\}$. The set of transitions is defined as follows:

$$T = \{Y \xrightarrow{a} \mathrm{next}(Z) \mid Y \in Q,\ a \in \Sigma_Z \text{ and } Z \in \mathrm{Red}(Y)\}$$

For each $\alpha \in \mathsf{U}(\varphi)$, we define the acceptance set T_α:

$$T_\alpha = \{Y \xrightarrow{a} \mathrm{next}(Z) \mid Y \in Q,\ a \in \Sigma_Z \text{ and } Z \in \mathrm{Red}_\alpha(Y)\}$$

Since $\emptyset \in Q$ and $\Sigma_\emptyset = \Sigma$, the transition $\emptyset \xrightarrow{\Sigma} \emptyset$ belongs to T and to T_α for each $\alpha \in \mathsf{U}(\varphi)$. Note that, if φ does not have *until* subformulae then there are no acceptance conditions, which means that all infinite paths are successful.

In practice, we only compute and include in $\mathcal{A}_\varphi$ the states and transitions that are reachable from the initial state $\{\varphi\}$ so that Q is only a subset of $2^{\mathrm{sub}(\varphi)}$. The first automaton in Figure 1.7 shows the complete construction, including the ε-transitions and the intermediary dashed states, for the response formula $\varphi = \mathsf{G}(p \rightarrow \mathsf{F}\, q)$. After removing the intermediary dashed states and the ε-transitions, we obtain the second automaton in Figure 1.7 where the transitions from the unique acceptance condition $T_{\mathsf{F}\, q}$ are labelled with $\mathsf{F}\, q$. As $\Sigma_{\neg p} \subseteq \Sigma$, the loop labeled $\Sigma_{\neg p}$ on the second state is redundant[‡]. Similarly, the transitions labeled $\Sigma_{\neg p \wedge q}$ is redundant. We obtain the third GBA $\mathcal{A}_\varphi$ in Figure 1.7. It is then easy to check that $\mathcal{L}(\mathcal{A}_\varphi) = \{u \in \Sigma^\omega \mid u \models \varphi\}$.

1.3.5. *Correctness*

More examples will conclude this section. Let us first show the correctness of the construction. The main result is stated in Theorem 1.2.

Theorem 1.2. *The automaton $\mathcal{A}_\varphi$ accepts precisely the models of φ, i.e.,*

$$\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi) = \{u \in \Sigma^\omega \mid u \models \varphi\}$$

In order to be precise, φ satisfies much more models by considering the larger set of propositional variables that do not appear in φ. Indeed such propositional variables are simply irrelevant for the satisfaction of φ. The proof of this theorem requires several lemmas and propositions. The first lemma is trivial.

Lemma 1.3. *Let Z be a* consistent *and* reduced *set of formulae in NNF. Let $u = a_0 a_1 a_2 \cdots \in \Sigma^\omega$ and $n \geq 0$. Then $u, n \models \bigwedge Z$ if and only if $u, n+1 \models \bigwedge \mathrm{next}(Z)$ and $a_n \in \Sigma_Z$.*

Using the equivalence $\bigwedge Y \equiv \bigvee_{Z \in \mathrm{Red}(Y)} \bigwedge Z$ we prove now:

Lemma 1.4. *Let Y be a subset of formulae in NNF and let $u \in \Sigma^\omega$ be an infinite word. If $u \models \bigwedge Y$ then there is $Z \in \mathrm{Red}(Y)$ such that $u \models \bigwedge Z$ and for every $\alpha = \alpha_1 \,\mathsf{U}\, \alpha_2 \in \mathsf{U}(\varphi)$, if $u \models \alpha_2$, then $Z \in \mathrm{Red}_\alpha(Y)$.*

Proof. Consider again the reduction rules presented in Table 1.1. At each step, either we have a single ε-transition $Y \xrightarrow{\varepsilon} Y_1$ and $\bigwedge Y \equiv \bigwedge Y_1$ or we have two ε-transitions $Y \xrightarrow{\varepsilon} Y_1$ and $Y \xrightarrow{\varepsilon} Y_2$ and $\bigwedge Y \equiv \bigwedge Y_1 \vee \bigwedge Y_2$. So there is a reduction path from Y to some $Z \in \mathrm{Red}(Y)$ such that $u \models \bigwedge Z$ and whenever we reduce an until formula $\alpha = \alpha_1 \,\mathsf{U}\, \alpha_2$ with $u \models \alpha_2$ we take the first reduction $Y' \xrightarrow{\varepsilon} Y' \setminus \{\alpha\} \cup \{\alpha_2\}$.

[‡]Actually this corresponds to a set of transitions which is contained in the set of transitions described by the loop labeled Σ.

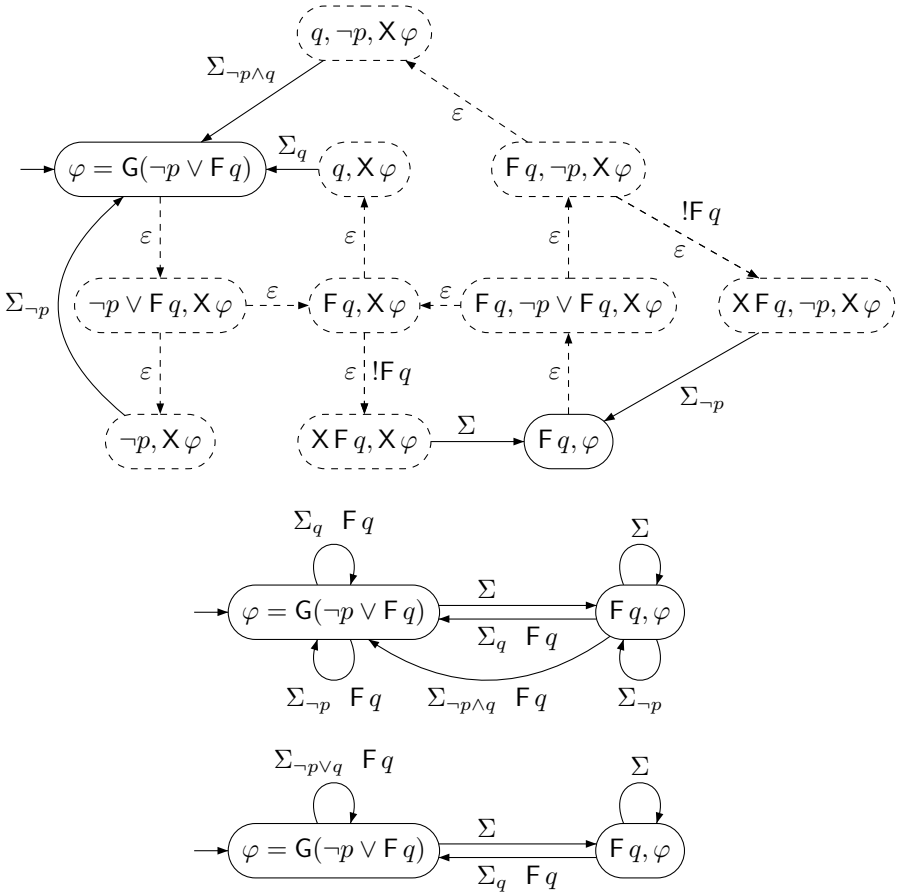


Fig. 1.7. GBA for the response formula

Now, let $\alpha = \alpha_1 \,\mathsf{U}\, \alpha_2 \in \mathsf{U}(\varphi)$ be such that $u \models \alpha_2$. Either α is never reduced along this path and indeed $Z \in \mathrm{Red}_\alpha(Y)$ or α is reduced and by the hypothesis above we took the unmarked ε-transition. Hence $Z \in \mathrm{Red}_\alpha(Y)$. □

Proposition 1.5. $\mathcal{L}(\varphi) \subseteq \mathcal{L}(\mathcal{A}_\varphi)$.

Proof. Let $u = a_0 a_1 a_2 \cdots \in \Sigma^\omega$ be such that $u \models \varphi$. By induction, we build

$$\rho = Y_0 \xrightarrow{a_0} Y_1 \xrightarrow{a_1} Y_2 \cdots$$

of $\mathcal{A}_\varphi$ such that for all $n \geq 0$ we have $u, n \models \bigwedge Y_n$ and there is some $Z_n \in \mathrm{Red}(Y_n)$ with $a_n \in \Sigma_{Z_n}$ and $Y_{n+1} = \mathrm{next}(Z_n)$. We start with $Y_0 = \{\varphi\}$. Assume now that $u, n \models \bigwedge Y_n$ for some $n \geq 0$. By Lemma 1.4, there is $Z_n \in \mathrm{Red}(Y_n)$ such that $u, n \models \bigwedge Z_n$ and for all until subformulae $\alpha = \alpha_1 \,\mathsf{U}\, \alpha_2 \in \mathsf{U}(\varphi)$, if $u, n \models \alpha_2$ then $Z_n \in \mathrm{Red}_\alpha(Y_n)$. Then we define $Y_{n+1} = \mathrm{next}(Z_n)$. Since $u, n \models \bigwedge Z_n$, Lemma 1.3 implies $a_n \in \Sigma_{Z_n}$ and $u, n+1 \models \bigwedge Y_{n+1}$. Therefore, ρ is a run for u in $\mathcal{A}_\varphi$.

It remains to show that ρ is successful. By definition, it starts from the initial state $\{\varphi\}$. Now let $\alpha = \alpha_1 \,\mathsf{U}\, \alpha_2 \in \mathsf{U}(\varphi)$. Assume there exists $N \geq 0$ such that $Y_n \xrightarrow{a_n} Y_{n+1} \notin T_\alpha$ for all $n \geq N$. Then $Z_n \notin \mathrm{Red}_\alpha(Y_n)$ for all $n \geq N$ and we deduce that $u, n \not\models \alpha_2$ for all $n \geq N$. But, since $Z_N \notin \mathrm{Red}_\alpha(Y_N)$, the formula α has been reduced using an ε-transition marked $!\alpha$ along the path from Y_N to Z_N. Therefore, $\mathsf{X}\alpha \in Z_N$ and $\alpha \in Y_{N+1}$. By construction of the run we have $u, N+1 \models \bigwedge Y_{N+1}$. Hence, $u, N+1 \models \alpha$, a contradiction with $u, n \not\models \alpha_2$ for all $n \geq N$. Consequently, the run ρ is successful and u is accepted by $\mathcal{A}_\varphi$. □

We prove now the converse inclusion.

Proposition 1.6. $\mathcal{L}(\mathcal{A}_\varphi) \subseteq \mathcal{L}(\varphi)$.

Proof. Let $u = a_0 a_1 a_2 \cdots \in \Sigma^\omega$ and let

$$\rho = Y_0 \xrightarrow{a_0} Y_1 \xrightarrow{a_1} Y_2 \cdots$$

be an accepting run of $\mathcal{A}_\varphi$ for the word u. We show by induction that

> for all $\psi \in \mathrm{sub}(\varphi)$ and $n \geq 0$, for all reduction path $Y_n \xrightarrow[*]{\varepsilon} Y \xrightarrow[*]{\varepsilon} Z$ with $a_n \in \Sigma_Z$ and $Y_{n+1} = \mathrm{next}(Z)$, if $\psi \in Y$ then $u, n \models \psi$.

The induction is on the formula ψ with the subformula ordering.

If $\psi = \top$ then the result is trivial. Assume next that $\psi = p \in \mathrm{AP}(\varphi)$. Since p is reduced, we have $p \in Z$ and it follows $\Sigma_Z \subseteq \Sigma_p$. Therefore, $p \in a_n$ and $u, n \models p$. The proof is similar if $\psi = \neg p$ for some $p \in \mathrm{AP}(\varphi)$.

If $\psi = \mathsf{X}\psi_1$ then $\psi \in Z$ and $\psi_1 \in Y_{n+1}$. By induction we obtain $u, n+1 \models \psi_1$ and we deduce $u, n \models \mathsf{X}\psi_1 = \psi$.

Assume now that $\psi = \psi_1 \wedge \psi_2$. Along the path $Y \xrightarrow[*]{\varepsilon} Z$ the formula ψ must be reduced so $Y \xrightarrow[*]{\varepsilon} Y' \xrightarrow[*]{\varepsilon} Z$ with $\psi_1, \psi_2 \in Y'$. By induction, we obtain $u, n \models \psi_1$ and $u, n \models \psi_2$. Hence, $u, n \models \psi$. The proof is similar for $\psi = \psi_1 \vee \psi_2$.

Assume next that $\psi = \psi_1 \,\mathsf{U}\, \psi_2$. Along the path $Y \xrightarrow[*]{\varepsilon} Z$ the formula ψ must be reduced so $Y \xrightarrow[*]{\varepsilon} Y' \xrightarrow{\varepsilon} Y'' \xrightarrow[*]{\varepsilon} Z$ with either $Y'' = Y' \setminus \{\psi\} \cup \{\psi_2\}$ or $Y'' = Y' \setminus \{\psi\} \cup \{\psi_1, \mathsf{X}\psi\}$. In the first case, we obtain by induction $u, n \models \psi_2$ and therefore $u, n \models \psi$. In the second case, we obtain by induction $u, n \models \psi_1$. Since $\mathsf{X}\psi$ is reduced we get $\mathsf{X}\psi \in Z$ and $\psi \in \mathrm{next}(Z) = Y_{n+1}$.

Let $k > n$ be minimal such that $Y_k \xrightarrow{a_k} Y_{k+1} \in T_\psi$ (such a value k exists since ρ is accepting). We first show by induction that $u, i \models \psi_1$ and $\psi \in Y_{i+1}$ for all $n < i < k$. Recall that $\psi \in Y_{n+1}$. So let $n < i < k$ be such that $\psi \in Y_i$. Let $Z' \in \mathrm{Red}(Y_i)$ be such that $a_i \in \Sigma_{Z'}$ and $Y_{i+1} = \mathrm{next}(Z')$. Since k is minimal we know that $Z' \notin \mathrm{Red}_\psi(Y_i)$. Hence, along any reduction path from Y_i to Z' we must use a step $Y' \xrightarrow[!\psi]{\varepsilon} Y' \setminus \{\psi\} \cup \{\psi_1, \mathsf{X}\psi\}$. By induction on the formula we obtain $u, i \models \psi_1$. Also, since $\mathsf{X}\psi$ is reduced, we have $\mathsf{X}\psi \in Z'$ and $\psi \in \mathrm{next}(Z') = Y_{i+1}$.

Second, we show that $u, k \models \psi_2$. Since $Y_k \xrightarrow{a_k} Y_{k+1} \in T_\psi$, we find some $Z' \in \mathrm{Red}_\psi(Y_k)$ such that $a_k \in \Sigma_{Z'}$ and $Y_{k+1} = \mathrm{next}(Z')$. Since $\psi \in Y_k$, along some reduction path from Y_k to Z' we use a step $Y' \xrightarrow{\varepsilon} Y' \setminus \{\psi\} \cup \{\psi_2\}$. By induction we obtain $u, k \models \psi_2$. Finally, we have shown $u, n \models \psi_1 \,\mathsf{U}\, \psi_2 = \psi$.

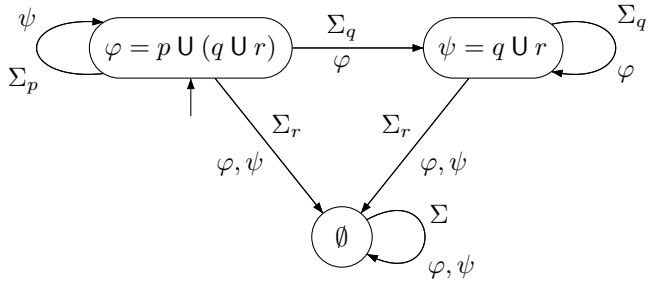


Fig. 1.8. GBA for nested until

The last case is when $\psi = \psi_1 \,\mathsf{R}\, \psi_2$. Along the path $Y \xrightarrow[*]{\varepsilon} Z$ the formula ψ must be reduced so $Y \xrightarrow[*]{\varepsilon} Y' \xrightarrow{} Y'' \xrightarrow[*]{\varepsilon} Z$ with either $Y'' = Y' \setminus \{\psi\} \cup \{\psi_1, \psi_2\}$ or $Y'' = Y' \setminus \{\psi\} \cup \{\psi_2, \mathsf{X}\,\psi\}$. In the first case, we obtain by induction $u, n \models \psi_1$ and $u, n \models \psi_2$. Hence, $u, n \models \psi$ and we are done. In the second case, we obtain by induction $u, n \models \psi_2$ and we get also $\psi \in Y_{n+1}$. Continuing with the same reasoning, we deduce easily that either $u, n \models \mathsf{G}\,\psi_2$ or $u, n \models \psi_2 \,\mathsf{U}\, (\psi_1 \wedge \psi_2)$. □

We have proved the correctness of our construction. We give now more examples and discuss simplifications that may be applied during the construction. First, consider $\varphi = p \,\mathsf{U}\, (q \,\mathsf{U}\, r)$. Here we have two until formulae, φ itself and $\psi = q \,\mathsf{U}\, r$, hence the GBA will have two acceptance sets T_φ and T_ψ. We can easily check that

$$\begin{aligned} \mathrm{Red}(\{\varphi\}) &= \{\{p, \mathsf{X}\,\varphi\}, \{q, \mathsf{X}\,\psi\}, \{r\}\} \\ \mathrm{Red}_\varphi(\{\varphi\}) &= \{\{q, \mathsf{X}\,\psi\}, \{r\}\} \\ \mathrm{Red}_\psi(\{\varphi\}) &= \{\{p, \mathsf{X}\,\varphi\}, \{r\}\} \end{aligned}$$

Hence, starting from the initial state $\{\varphi\}$, the construction introduces two new states $\{\psi\}$ and $\emptyset$. We compute

$$\begin{aligned} \mathrm{Red}(\{\psi\}) &= \{\{q, \mathsf{X}\,\psi\}, \{r\}\} \\ \mathrm{Red}_\varphi(\{\psi\}) &= \{\{q, \mathsf{X}\,\psi\}, \{r\}\} \\ \mathrm{Red}_\psi(\{\psi\}) &= \{\{r\}\} \end{aligned}$$

There are no new states, so the construction terminates and we obtain the GBA of Figure 1.8 where the transitions from T_φ and T_ψ are marked φ and ψ respectively.

The polynomial space upper bound for LTL model-checking can be then stated as follows.

Proposition 1.7. *[41] Given a finite and total Kripke structure $\mathcal{M}$, a state s in $\mathcal{M}$ and an LTL formula φ, it is possible to check in space polynomial in $|\varphi| + log|\mathcal{M}|$ whether $\mathcal{M}, s \models_\forall \varphi$ and $\mathcal{M}, s \models_\exists \varphi$.*

Indeed, $\mathcal{M}, s \models_\forall \varphi$ holds if and only if $\mathrm{L}(\mathcal{A}_{\mathcal{M},s} \otimes \mathcal{A}_{\neg\varphi}) = \emptyset$ where

- $\mathcal{A}_{\mathcal{M},s}$ is the obvious Büchi automaton of size $\mathcal{O}(|\mathcal{M}|)$ such that $\mathcal{L}(\mathcal{A}_{\mathcal{M},s}) = \lambda\mathrm{Paths}(\mathcal{M}, s)$ (all states are accepting),

- $\mathcal{A}_{\neg\varphi}$ is the Büchi automaton recognizing the models for $\neg\varphi$ obtained with the previous constructions. Its size is $2^{\mathcal{O}(|\varphi|)}$.
- "$\otimes$" denotes the product operation used for intersection.

Nonemptiness of $\mathcal{A}_{\mathcal{M},s} \otimes \mathcal{A}_{\neg\varphi}$ can be then checked on the fly in nondeterministic polynomial space since $\mathcal{A}_{\mathcal{M},s} \otimes \mathcal{A}_{\neg\varphi}$ is of size $|\mathcal{M}| \times 2^{\mathcal{O}(|\varphi|)}$. By Savitch's theorem (see e.g. [47]), we then obtain Proposition 1.7. A similar reasoning can be done for existential model checking since $\mathcal{M}, s \models_\exists \varphi$ holds if and only if $\mathrm{L}(\mathcal{A}_{\mathcal{M},s} \otimes \mathcal{A}_\varphi) \neq \emptyset$. Furthermore, the properties about the construction of $\mathcal{A}_\varphi$ allow also to get the polynomial space upper bound for satisfiability and validity.

Proposition 1.8. *[35] Checking whether an LTL formula φ is satisfiable (or valid) can be done in space polynomial in $|\varphi|$.*

1.3.6. *On the fly simplifications of the GBA*

One optimization was already included in the contruction: when reducing a set Y of formulae, we start with *maximal* formulae. This strategy produces fewer ε-transitions and fewer states in the set $\mathrm{Red}(Y)$. For instance, assume that $Y = \{\varphi, \mathsf{G}\,\varphi\}$. If we reduce first the maximal formula $\mathsf{G}\,\varphi$ we obtain $Y' = \{\varphi, \mathsf{X}\,\mathsf{G}\,\varphi\}$ and it remains to reduce φ. We obtain the sets $Z \cup \{\mathsf{X}\,\mathsf{G}\,\varphi\}$ for $Z \in \mathrm{Red}(\{\varphi\})$. If instead we start by reducing φ we obtain the sets $Z \cup \{\mathsf{G}\,\varphi\}$ for $Z \in \mathrm{Red}(\{\varphi\}$. But then $\mathsf{G}\,\varphi$ has to be reduced so that we obtain the sets $Z \cup \{\varphi, \mathsf{X}\,\mathsf{G}\,\varphi\}$ and φ has to be reduced again. We obtain finally sets $Z_1 \cup Z_2 \cup \{\mathsf{X}\,\mathsf{G}\,\varphi\}$ for $Z_1, Z_2 \in \mathrm{Red}(\{\varphi\})$.

Next, during the reduction, we may replace a set Y by Y' provided they are *equivalent*, i.e., $\bigwedge Y \equiv \bigwedge Y'$. Checking equivalence is as hard as constructing the automaton, so we only use easy syntactic equivalences. For instance, we may use the following rules:

$$\begin{aligned}
&\text{If } \psi = \psi_1 \vee \psi_2 \text{ and } \psi_1 \in Y \text{ or } \psi_2 \in Y: && Y \xrightarrow{\varepsilon} Y \setminus \{\psi\}\\
&\text{If } \psi = \psi_1 \,\mathsf{U}\, \psi_2 \text{ and } \psi_2 \in Y: && Y \xrightarrow{\varepsilon} Y \setminus \{\psi\}\\
&\text{If } \psi = \psi_1 \,\mathsf{R}\, \psi_2 \text{ and } \psi_1 \in Y: && Y \xrightarrow{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2\}
\end{aligned}$$

We explain now an easy and useful simplification of the constructed GBA: when two states have the same outgoing transitions, then they can be merged. More precisely, two states s_1 and s_2 of a GBA $\mathcal{A} = (Q, \Sigma, I, T, T_1, \ldots, T_n)$ have the same outgoing transitions if for all $a \in \Sigma$ and $s \in Q$, we have

$$\begin{aligned}
&(s_1, a, s) \in T \iff (s_2, a, s) \in T\\
\text{and}\quad &(s_1, a, s) \in T_i \iff (s_2, a, s) \in T_i \qquad \text{for all } 1 \leq i \leq n.
\end{aligned}$$

In this case, the two states s_1 and s_2 can be merged without changing the accepted language. When merging these states, we redirect all transitions to either s_1 or s_2 to the new merged state $s_{1,2}$.

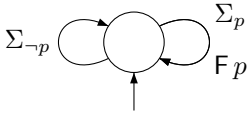
With our construction, we have an easy sufficient condition ensuring that two states Y and Y' have the same outgoing transitions:

$$\begin{cases} \mathrm{Red}(Y) = \mathrm{Red}(Y') & \text{and} \\ \mathrm{Red}_\alpha(Y) = \mathrm{Red}_\alpha(Y') & \text{for all } \alpha \in \mathsf{U}(\varphi) \end{cases} \tag{1.1}$$

For instance, consider the formula $\varphi = \mathsf{G}\,\mathsf{F}\,p$. We have

$$\begin{aligned} \mathrm{Red}(\{\varphi\}) &= \{\{p, \mathsf{X}\,\varphi\}, \{\mathsf{X}\,\mathsf{F}\,p, \mathsf{X}\,\varphi\}\} \\ \mathrm{Red}_{\mathsf{F}\,p}(\{\varphi\}) &= \{\{p, \mathsf{X}\,\varphi\}\} \end{aligned}$$

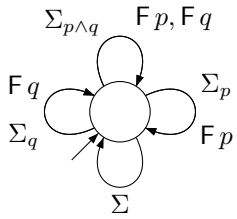
Hence, from the initial state $\{\varphi\}$, we reach a new state $\{\mathsf{F}\,p, \varphi\}$. We can easily check that the two states satisfy (1.1) hence they can be merged and the resulting GBA has only one state and two transitions:



Similarly, if we consider $\varphi = \mathsf{G}\,\mathsf{F}\,p \wedge \mathsf{G}\,\mathsf{F}\,q$. We have

$$\begin{aligned} \mathrm{Red}(\{\varphi\}) = \{&\{p, q, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,q\}, \{p, \mathsf{X}\,\mathsf{F}\,q, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,q\}, \\ &\{q, \mathsf{X}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,q\}, \{\mathsf{X}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{F}\,q, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,q\}\} \\ \mathrm{Red}_{\mathsf{F}\,p}(\{\varphi\}) = \{&\{p, q, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,q\}, \{p, \mathsf{X}\,\mathsf{F}\,q, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,q\}\} \\ \mathrm{Red}_{\mathsf{F}\,q}(\{\varphi\}) = \{&\{p, q, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,q\}, \{q, \mathsf{X}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,q\}\} \end{aligned}$$

Hence, a direct application of the construction produces an automaton with 4 new states. However, we can easily check with (1.1) that all states have the same outgoing transitions. Hence, again, the resulting GBA has only one state and 4 transitions which are marked $\mathsf{F}\,p$ or $\mathsf{F}\,q$ if they belong to $T_{\mathsf{F}\,p}$ or $T_{\mathsf{F}\,q}$:



More examples are given in Figure 1.9.

Other optimizations can be found in [14, 16], as well as simplifications of Büchi automata generated from LTL formulae in [17].

1.3.7. *Related work*

The construction presented in this section is in the same vein as those presented in [12, 14, 16, 48], see also [20]. For instance, the states of the automata in [16]

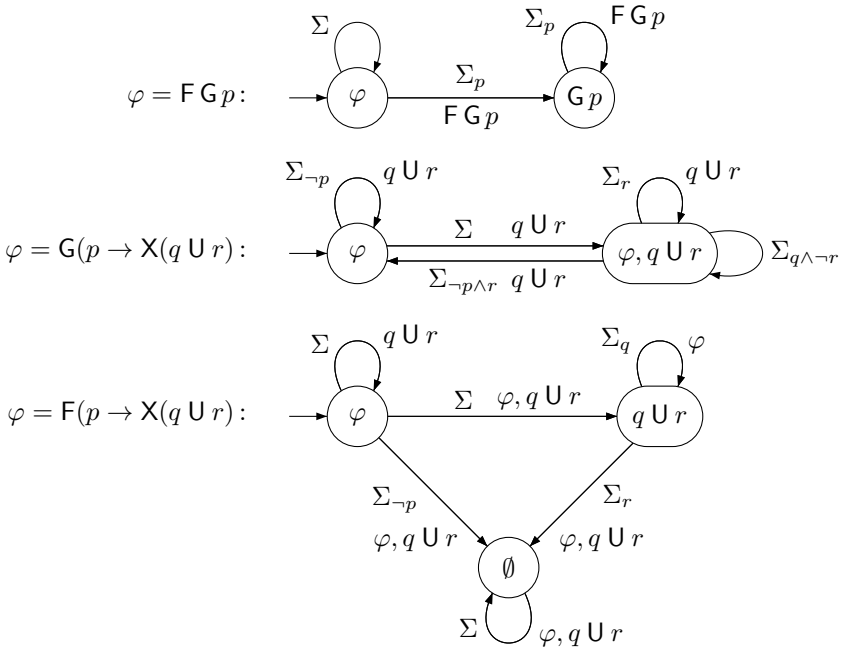


Fig. 1.9. More examples

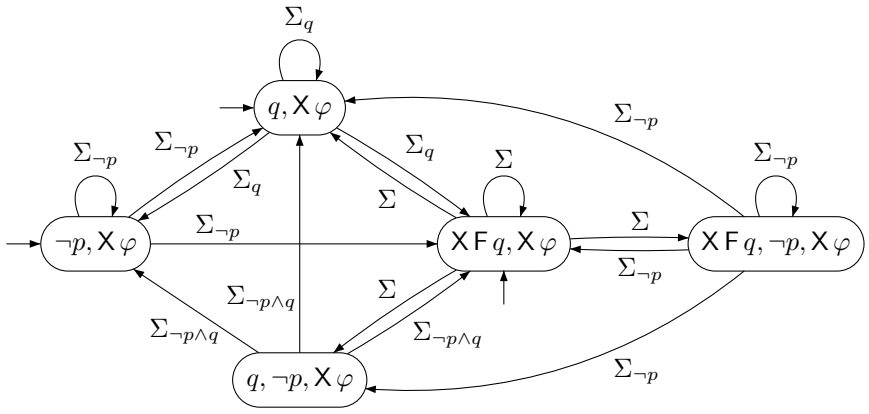


Fig. 1.10. Another GBA for the response formula

correspond to our reduced sets and the transitions are constructed by applying first the *next* step and then the *reduction* phase. For the response formula $\mathsf{G}(p \to \mathsf{F}\,q)$ there are 5 reduced sets (see Figure 1.7) and we would get the automaton of Figure 1.10. Note that, except possibly for the initial state, each of our states is the next state of some reduced set. But we usually have several reduced sets having the same next set. Hence, our construction cannot yield more states (still apart

possibly for the initial state) and usually yield fewer states than the construction of [16] as in the example above. Another difference is that [16] uses acceptance conditions based on states whereas we use transition-based acceptance.

Moreover, tableau methods for LTL, see e.g. [15], contain decomposition rules similar to the reduction rules and contain an additional step to check global conditions about eventuality formulae (in $\mathsf{U}(\varphi)$). Our construction is therefore similar to such methods since the expansion of a state (or branch in the tableau terminology) is done on demand and the verification of eventualities is simply performed by transition-based acceptance conditions. Hence, the two phases of the method in [15] apply also herein and in the worst-case we also obtain exponential-size automata. It is worth recalling that a *one-pass* tableaux calculus for LTL is presented in [49] by using additional control structures (no step to check global conditions). Finally, there is another solution to encode the second phase of the method in [15], which is to translate LTL model-checking into CTL model-checking with fairness conditions [50].

Helpful bibliographical remarks can also be found at the end of [48, Chapter 5] as well as in [20].

1.4. Extensions

In this section, we consider three extensions for developments made in Section 1.3. Firstly, we show how a procedure solving LTL model-checking can be used to solve CTL* model-checking. Secondly, we present an extension of LTL with temporal operators defined from finite-state automata. Thirdly, we show that adding a single temporal operator defined by a context-free language leads to undecidability.

1.4.1. *Model-checking for branching-time* CTL*

Even though CTL* is a branching-time logic, $\mathrm{MC}^{\forall}(\mathrm{CTL}^*)$ can be solved by using as subroutine the algorithm for LTL model-checking with a simple renaming technique.

Proposition 1.9. *[34]* $\mathrm{MC}^{\forall}(\mathrm{CTL}^*)$ *is* PSPACE*-complete.*

Proof. Since $\mathrm{MC}^{\forall}(\mathrm{LTL})$ is a subproblem of $\mathrm{MC}^{\forall}(\mathrm{CTL}^*)$, the PSPACE-hardness is immediate. In order to show that $\mathrm{MC}^{\forall}(\mathrm{CTL}^*)$ is in PSPACE, we use known techniques for LTL plus renaming.

For each quantifier $\mathcal{Q} \in \{\exists, \forall\}$, we write $\mathrm{MC}^{\mathcal{Q}}_{\mathrm{LTL}}(\mathcal{M}, s, \varphi)$ to denote the function that returns `true` if and only if $\mathcal{M}, s \models_{\mathcal{Q}} \varphi$. We have seen that these functions can be computed in polynomial space in $|\varphi| + log(|\mathcal{M}|)$ (Proposition 1.7). In order to establish the PSPACE upper bound, here is an algorithm based on formulae renaming using only polynomial space:

$\mathrm{MC}^{\forall}_{\mathrm{CTL}^*}(\mathcal{M} = \langle W, R, \lambda \rangle, s \in W, \varphi)$

- If E, A do not occur in φ, then return $\mathrm{MC}^{\forall}_{\mathrm{LTL}}(\mathcal{M}, s, \varphi)$.
- Otherwise φ contains a subformula $\mathcal{Q}\psi$ where E, A do not occur in ψ and $\mathcal{Q} \in \{\mathsf{E}, \mathsf{A}\}$. This means that the formula ψ belongs to LTL. Let $\mathcal{Q}'$ be "$\forall$" if $\mathcal{Q} = \mathsf{A}$, "$\exists$" otherwise. Let $p_{\mathcal{Q}\psi}$ be a new propositional variable. We define λ' an extension of λ for every $s' \in W$ by:

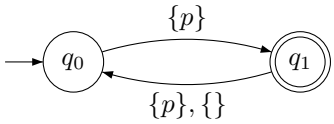
$$\lambda'(s') = \begin{cases} \lambda(s') \cup \{p_{\mathcal{Q}\psi}\} & \text{if } \mathrm{MC}^{\mathcal{Q}'}_{\mathrm{LTL}}(\mathcal{M}, s', \psi) \\ \lambda(s') & \text{otherwise.} \end{cases}$$

  Return $\mathrm{MC}^{\forall}_{\mathrm{CTL}^*}(\langle W, R, \lambda' \rangle, s, \varphi[\mathcal{Q}\psi \leftarrow p_{\mathcal{Q}\psi}])$ where $\varphi[\mathcal{Q}\psi \leftarrow p_{\mathcal{Q}\psi}]$ is obtained from φ by replacing every occurrence of $\mathcal{Q}\psi$ by $p_{\mathcal{Q}\psi}$.

Since in $\mathrm{MC}^{\forall}_{\mathrm{CTL}^*}(\mathcal{M}, s, \varphi)$, the recursion depth is at most $|\varphi|$, we can show that $\mathrm{MC}^{\forall}_{\mathrm{CTL}^*}$ uses only polynomial space since $\mathrm{MC}^{\exists}_{\mathrm{LTL}}$ and $\mathrm{MC}^{\forall}_{\mathrm{LTL}}$ require only polynomial space. The soundness of the algorithm is not very difficult to show. □

1.4.2. *Automata-based temporal operators*

We have seen how to build a GBA $\mathcal{A}_\varphi$ such that $\mathcal{L}(\mathcal{A}_\varphi)$ is equal to the set of models for φ (in LTL). However, it is known that LTL is strictly less expressive than Büchi automata [21]. It is not always easy to figure out whether a given Büchi automaton on the alphabet $2^{\mathrm{AP}(\varphi)}$ corresponds to an LTL formula where $\mathrm{AP}(\varphi)$ denotes the set of propositional variables occurring in φ. For instance, is there an LTL formula, counterpart of the automaton presented below?



We invite the reader to analyze why none of the formulae below is adequate:

(1) $p \wedge \mathsf{X}\neg p \wedge \mathsf{G}(p \leftrightarrow \mathsf{X}\mathsf{X}p)$,
(2) $p \wedge \mathsf{G}(p \rightarrow \mathsf{X}\mathsf{X}p)$,
(3) $q \wedge \mathsf{X}\neg q \wedge \mathsf{G}(q \leftrightarrow \mathsf{X}\mathsf{X}q) \wedge \mathsf{G}(q \rightarrow p)$.

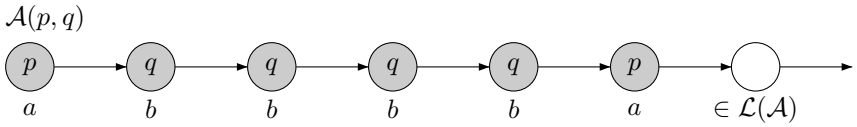
For instance, formula (1) defines a unique model over the two-letters alphabet $2^{\{p\}}$. Note that formula (3) requires that p holds at all even positions but uses an extra propositional variable which is not allowed by the alphabet of the automaton. Actually, there exist quite simple properties that cannot be expressed with LTL.

To check whether an ω-regular language L can be expressed in LTL, one may compute the *syntactic monoid* of L from the automaton which recognizes L and check that this monoid is *aperiodic*, see e.g. the survey chapter [51]. This procedure can be applied to prove the following result (even though the proof in [21] is different).

Proposition 1.10. *[21] There is no* LTL *formula φ built over the unique propositional variable p such that $\mathcal{L}(\varphi)$ is exactly the set of* LTL *models such that p holds on every even position (on odd positions, p may be true or not).*

That is why, in [21], an extension of LTL has been introduced by adding temporal operators defined with finite-state automata. Alternatively, right-linear grammars can also be used to define regular languages. Let $\mathcal{A} = \langle \Sigma, S, S_0, \rho, F \rangle$ be a finite-state automaton with letters from a linearly ordered alphabet Σ, say with the ordering $a_1 < \ldots < a_k$. Assume that we already have defined the formulae $\varphi_1, \ldots, \varphi_k$. Then, $\mathcal{A}(\varphi_1, \ldots, \varphi_k)$ is a new formula in the *Extended Temporal Logic* (ETL). The relation $u, i \models \mathcal{A}(\varphi_1, \ldots, \varphi_k)$ holds when a finite pattern induced from $\mathcal{L}(\mathcal{A})$ exists from position i. There is a correspondence between the letters $a_1, \ldots, a_k$ and the arguments $\varphi_1, \ldots, \varphi_k$. More precisely $u, i \models \mathcal{A}(\varphi_1, \ldots, \varphi_k)$ if there is a finite word $a_{i_1} a_{i_2} \ldots a_{i_n} \in \mathcal{L}(\mathcal{A})$ such that for every $1 \leq j \leq n$, we have $u, i + (j-1) \models \varphi_{i_j}$.

Note that, if $S_0 \cap F \neq \emptyset$, then $\varepsilon \in \mathcal{L}(\mathcal{A})$ and $\mathcal{A}(\varphi_1, \ldots, \varphi_k)$ is equivalent to $\top$. Observe also that in the condition above, the index of the k-th letter (with $k \in \{1, \ldots, n\}$) determines which argument must hold at the $(k-1)$-th next position. We present below a model for the ETL formula $\mathcal{A}(p, q)$ with $\mathcal{L}(\mathcal{A}) = \{ab^i a \mid i \geq 0\}$ and $a < b$.



By way of example, the formula $\varphi \mathsf{U} \psi$ is equivalent to $\mathcal{B}(\varphi, \psi)$ with $\mathcal{L}(\mathcal{B}) = a^*b$ and $a < b$. Similarly, the *weakness* of LTL described in Proposition 1.10 can be fixed within ETL: the formula $\neg\mathcal{A}(\top, \neg p)$ with $\mathcal{L}(\mathcal{A}) = (a^2)^*b$ holds exactly in models such that the propositional variable p holds on every even position.

Formally, the syntax of ETL allows the propositional variables, the boolean connectives and temporal modalities of the form $\mathcal{A}(\varphi_1, \ldots, \varphi_k)$. Note that it does not include the temporal operators from LTL since they can be expressed with automata.

In order to illustrate the expressive power of ETL, it is sufficient to consider the fragment ETL$^-$ defined by the syntax below:

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \vee \varphi \mid K \cdot \varphi$$

where $K \subseteq \Sigma^*$ is a regular language of finite words over some finite alphabet $\Sigma \subseteq 2^{\mathrm{AP}}$ such that each letter $a \in \Sigma$ is a finite set. The semantics is that $w \models K \cdot \varphi$ if we can write $w = uv$ with $u \in K$ and $v \models \varphi$. To show that $K \cdot \varphi$ can be expressed in ETL, consider an automaton $\mathcal{A}$ for $K \cdot \#$ where $\#$ is a new letter (larger than all letters from Σ). For each $a \in \Sigma$, let $\varphi_a = \bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \neg p$. Then, $K \cdot \varphi$ is expressed by $\mathcal{A}((\varphi_a)_{a \in \Sigma}, \varphi)$ where φ_a is substituted for $a \in \Sigma$ and φ is substituted for the trailing $\#$.

Lemma 1.11. *For any ω-regular language L over a finite alphabet $\Sigma \subseteq 2^{\mathrm{AP}}$ made of finite letters, there is a formula φ in* ETL^- *such that $L = \mathcal{L}(\varphi)$.*

Proof. First, observe that any ω-regular language L over Σ can be written as a finite union of languages of the form $K \cdot L$ where $K \subseteq \Sigma^*$ is regular and $L \subseteq \Sigma^\omega$ is deterministic, i.e., recognized by a deterministic Büchi automaton. This can be easily derived from a deterministic Muller automaton recognizing L.

So consider a deterministic and complete Büchi automaton $\mathcal{A} = (Q, \Sigma, \{i\}, \delta, F)$. For each $s \in Q$, we define

$$M^s = \{u \in \Sigma^* \mid \delta(i,u) = s\}$$
$$N^s = \{v \in \Sigma^* \mid \delta(s,v) \in F\}.$$

Then, we can show

$$\overline{\mathcal{L}(\mathcal{A})} = \bigcup_{s \in Q} M^s \cdot \overline{N^s \cdot \Sigma^\omega}$$

We deduce immediately that $\mathcal{L}(\mathcal{A})$ can be expressed with the formula

$$\neg \bigvee_{s \in Q} M^s \cdot \neg(N^s \cdot \top)$$

Consequently, given an ω-regular language defined by a finite union of the form $\bigcup K_i \cdot L_i$ where each $K_i \subseteq \Sigma^*$ is regular, each $L_i \subseteq \Sigma^\omega$ is deterministic, the corresponding ETL^- formula is of the form below:

$$\bigvee_i K_i \cdot \left(\neg \bigvee_{s \in Q_i} M_i^s \cdot \neg(N_i^s \cdot \top) \right)$$

□

As a corollary, any ω-regular language can be defined by an expression obtained from the grammar $\mathcal{L} ::= \Sigma^\omega \mid \mathcal{L} \cup \mathcal{L} \mid \overline{\mathcal{L}} \mid K \cdot \mathcal{L}$, where K ranges over the regular languages in Σ^*.

Even though ETL formulae are seldom used in specification languages, its main theoretical assets rest on its high expressive power and on the relatively low complexity of satisfiability/model-checking problems as stated below.

Proposition 1.12. *[9]*

(I) $\mathrm{MC}^\forall(\mathrm{ETL})$, $\mathrm{MC}^\exists(\mathrm{ETL})$ *and* $\mathrm{SAT}(\mathrm{ETL})$ *are* PSPACE*-complete.*
(II) ETL *has the same expressive power as Büchi automata.*

An automata-based construction for ETL formulae can be found in [21], leading to Proposition 1.12(I). Proposition 1.12(II) is a corollary of Lemma 1.11.

Proposition 1.12(I) entails that ETL model-checking is not more difficult than LTL model-checking in the worst case, modulo logarithmic space many-one reductions. This is quite surprising in view of the expressive power of ETL – Proposition 1.12(II). Hence, the class of languages defined by ETL formulae is equal to the class of languages defined by

- Büchi automata (Proposition 1.12(II)),
- formulae from monadic second-order theory for $\langle \omega, < \rangle$ (S1S) and ω-regular expressions (finite unions of sets UV^ω with regular $U, V \subseteq \Sigma^*$), see e.g. [52, Chapter III],
- formulae from LTL with second-order quantification. In such an extension of LTL, we allow formulae of the form $\forall p \,.\, \varphi$ with $u, i \models \forall p \,.\, \varphi$ if for every u' such that u and u' agree on all propositional variables different from p, we have $u', i \models \varphi$, see e.g. [53].
- formulae from LTL with fixed-point operators [54].

So, ETL is a powerful extension of LTL but the above equivalences do not mean that all the above formalisms have the same conciseness. Actually, we know the following complexity results:

- the nonemptiness problem for Büchi automata is NLogSpace-complete,
- $\mathrm{MC}^\forall$(ETL), $\mathrm{MC}^\exists$(ETL) and SAT(ETL) are PSpace-complete,
- satisfiability for LTL with fixed-point operators is PSpace-complete [54],
- satisfiability for S1S is non-elementary (time complexity is not bounded by any tower of exponential of fixed height) [43].

So, S1S is the most concise language for describing ω-regular languages.

1.4.3. *Context-free extensions*

It is possible to extend the definition of ETL by replacing formulae of the form $\mathcal{A}(\varphi_1, \ldots, \varphi_n)$ by formulae of the form $\mathrm{L}(\varphi_1, \ldots, \varphi_n)$ where L is a language of finite words specified within a fixed formalism. The language L is again viewed as a set of patterns, not necessarily regular. For a class $\mathcal{C}$ of languages, we write PC[$\mathcal{C}$] to denote the extension the propositional calculus with formulae of the form $\mathrm{L}(\varphi_1, \ldots, \varphi_n)$ for some $\mathrm{L} \in \mathcal{C}$. Obviously, ETL is precisely equivalent to PC[REG] where REG is the class of regular languages represented by finite-state automata. We have seen that ETL is decidable and it is natural to wonder whether PC[CF] is also decidable where CF is the class of context-free languages (represented by context-free grammars).

1.4.3.1. *Undecidability of* PC[CF]

Since numerous problems for context-free languages are undecidable, it is not very surprising to get the following result.

Proposition 1.13. SAT(PC[CF]) *is undecidable.*

Before presenting the proof, let us recall that the next operator X and the until operator U (and the derived operators F and G) can be defined as operators obtained from finite-state automata. Hence, in the proof below, we use them freely.

Proof. We show that the validity problem for PC[CF] is undecidable, which entails the undecidability of SAT(PC[CF]) since PC[CF] is closed under negations. We reduce the universality problem for context-free grammars (see e.g. the textbook [55]) into the validity problem. Let G be a context-free grammar over the terminal alphabet $\Sigma = \{a_1, \ldots, a_n\}$. We write G^+ to denote the CF grammar over the terminal alphabet $\Sigma^+ = \{a_1, \ldots, a_n, a_{n+1}\}$ such that $\mathcal{L}(G^+) = \mathcal{L}(G) \cdot \{a_{n+1}\}$. The letter a_{n+1} is simply an end marker. G^+ can be effectively computed from G.

Let UNI be the formula

$$(\neg p_{n+1} \,\mathsf{U}\, (p_{n+1} \wedge \mathsf{X}\,\mathsf{G}\,\neg p_{n+1})) \wedge \mathsf{G} \bigvee_{1\leq i\leq n+1} \left(p_i \wedge \bigwedge_{1\leq j\leq n+1, j\neq i} \neg p_j \right)$$

The structures satisfying UNI are precisely those for which exactly one variable from $p_1, \ldots, p_{n+1}$ holds at each state and, p_{n+1} holds at a unique state of the model. Hence, we characterize structures that can be naturally viewed as finite words, possibly in $\mathcal{L}(G^+)$. We show that $\mathcal{L}(G) = \Sigma^*$ if and only if UNI $\rightarrow \mathcal{L}(G^+)$ is valid.

Indeed, if $\mathcal{L}(G) \neq \Sigma^*$, say $a_{i_1} a_{i_2} \cdots a_{i_\ell} \notin \mathcal{L}(G)$. Let u be the model $\{p_{i_1}\} \cdot \{p_{i_2}\} \cdots \{p_{i_\ell}\} \cdot \{p_{n+1}\} \cdot \{p_1\}^\omega$. We have $u \models \text{UNI} \wedge \neg\mathcal{L}(G^+)(p_1, \ldots, p_{n+1})$. So UNI $\rightarrow \mathcal{L}(G^+)$ is not valid. Conversely, it is easy to show that $\mathcal{L}(G) = \Sigma^*$ implies UNI $\rightarrow \mathcal{L}(G^+)$ is valid since every structure satisfying the formula UNI corresponds to a word in Σ^*. □

Proposition 1.13 is interesting, but after all, it rests on the fact that PC[CF] can easily encode universality for context-free grammars. It would be more interesting to establish that undecidability still holds for a very small fragment of CF, which is the subject of the next section.

1.4.3.2. *When a single context-free language leads to high undecidability*

The main result of this section is the (high) undecidability of the model-checking problem for PC[L_1] for the context-free language $\mathrm{L}_1 = \{a_1^k \cdot a_2 \cdot a_1^k \cdot a_3 \mid k \geq 0\}$. We start by introducing the auxiliary language $\mathrm{L}_0 = \{a_1^k \cdot a_2 \cdot a_1^{k-1} \cdot a_3 \mid k \geq 1\} = a_1 \cdot \mathrm{L}_1$. The next operator X, the eventuality operator F and the temporal operator defined from L_0 are definable in PC[L_1] thanks to the following equivalences:

- $\mathsf{X}\,\varphi \equiv \mathrm{L}_1(\bot, \top, \varphi)$,
- $\mathsf{F}\,\varphi \equiv \mathrm{L}_1(\top, \varphi, \top)$,
- $\mathrm{L}_0(\varphi_1, \varphi_2, \varphi_3) \equiv \varphi_1 \wedge \mathsf{X}\,\mathrm{L}_1(\varphi_1, \varphi_2, \varphi_3)$ since $\mathrm{L}_0 = a_1 \cdot \mathrm{L}_1$.

In the sequel, we therefore freely use these operators in PC[L_1] as well as the dual operator G.

Proposition 1.14. *Satisfiability for* PC[L_1] *is undecidable.*

Proof. We reduce the recurring domino problem DOMREC [56] to satisfiability for PC[L_1]. Let Sides $= \{\mathit{left}, \mathit{right}, \mathit{up}, \mathit{down}\}$ and recall that a domino game is a structure Dom $= \langle C, D, \gamma\rangle$ where C is a finite set of colours, D is a finite set of dominoes, and $\gamma : D \times \text{Sides} \to C$ is a map that assigns a color to each side of the dominoes. Dom can tile $\mathbb{N}\times\mathbb{N}$ if and only if there is a map $f : \mathbb{N}\times\mathbb{N} \to D$ that satisfies the color constraints. This means that only domino sides with identical colors can be adjacent (no rotation of dominoes is allowed). The problem DOMREC, known to be Σ^1_1-complete [56], takes as input a domino game Dom with a distinguished color c and asks whether Dom can pave $\mathbb{N}\times\mathbb{N}$ where the color c occurs infinitely often in the first column.

Let Dom $= \langle C, D, \gamma\rangle$ be a domino game with $C = \{1,\ldots,n\}$, $D = \{1,\ldots,m\}$, and $c = 1$. So we have an instance of DOMREC. Let us explain the syntactic ressources we shall need. We use the following propositional variables:

- in is a propositional variable that holds when the state encodes a position in $\mathbb{N}^2$. Indeed, there are states in the model that do not correspond to positions in $\mathbb{N}^2$. In order to facilitate the presentation, we also introduce out that is equivalent to the negation of in.
- For every $j \in D$, we introduce the variable d_j with intended meaning that "the position in $\mathbb{N}^2$ associated with the current state is occupied by a domino of type j".
- For every $i \in C$, we use the variables up_i, down_i, left_i, right_i. For instance, up_1 holds whenever the domino on the position associated with the current state has color 1 on its top.

Every state encoding a position in $\mathbb{N}^2$ is occupied by a unique domino:

$$\mathsf{G}\left(\mathit{in} \to \bigvee_{1\le j\le m}\left(d_j \wedge \bigwedge_{1\le k\le m, k\neq j} \neg d_k\right)\right)$$

Propositional variables for colours are compatible with the definition of domino types:

$$\bigwedge_{1\le j\le m}\ \bigwedge_{s\in\text{Sides}} \mathsf{G}\left(\mathit{in}\wedge d_j \to s_{\gamma(j,s)} \wedge \bigwedge_{1\le k\le n, k\neq\gamma(j,s)} \neg s_k\right)$$

We write PAVE to denote the conjunction of the above formulae. Now, we shall define the states of the model that correspond to positions in $\mathbb{N}^2$. We write SNAKE to denote the conjunction of the following formulae:

- $\mathsf{G}(\mathit{in} \leftrightarrow \neg\mathit{out})$,
- $\mathit{in} \wedge \mathsf{X}\,\mathit{out} \wedge \mathsf{XX}\,\mathit{in} \wedge \mathsf{XXX}\,\mathit{in} \wedge \mathsf{XXXX}\,\mathit{out}$,
- $\mathsf{G}(\mathit{out} \to \mathsf{X}\,\mathrm{L}_1(\mathit{in}, \mathit{out}, \mathit{in} \wedge \mathsf{X}\,\mathit{out}))$.

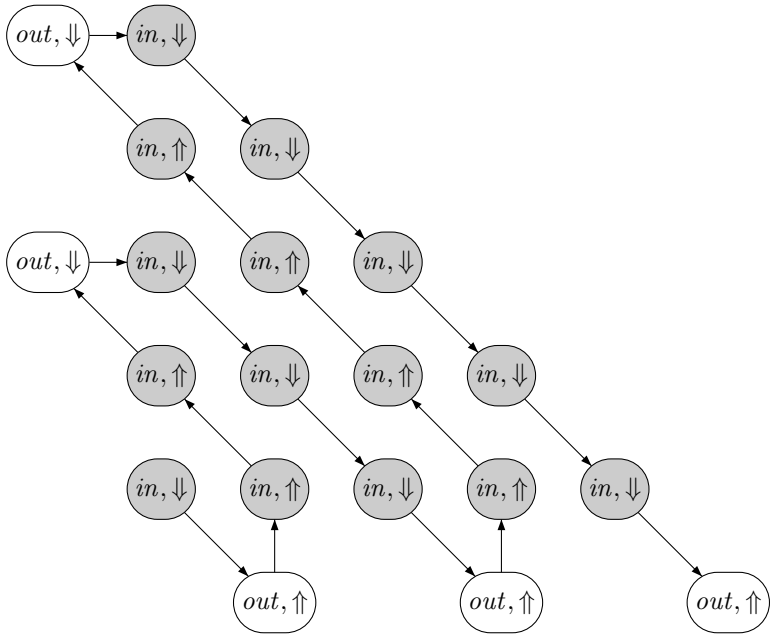


Fig. 1.11. The path in $\mathbb{N}^2$

The only structure (built over $\{in, out\}$) satisfying SNAKE is:

$$\{in\} \cdot \{out\} \cdot \{in\}^2 \cdot \{out\} \cdot \{in\}^3 \cdot \{out\} \cdot \{in\}^4 \ldots$$

This sequence makes reference to the path presented in Figure 1.11, where the set of grey nodes encodes $\mathbb{N}^2$. The difficulty of the proof is not to design a path through $\mathbb{N}^2$ but rather to define a path on which it is easy to access to neighbours (right or up). Moreover, we need to identify the positions encoding the first column of $\mathbb{N}^2$. Consequently, a bit more work is required.

For every state *in*, we need to remember if it occurs in a sequence of *in* that is upward or downward (alternately, this encodes the parity of i for each block $\{out\} \cdot \{in\}^i \cdot \{out\}$). Indeed, this criterion is relevant to access to (right or up) neighbours. We introduce the variables $\Uparrow$ and $\Downarrow$. The former one $\Uparrow$ is only introduced to facilitate the presentation. We write DIRECTION to denote the conjunction of the following formulae:

- $\mathsf{G}(\Uparrow \leftrightarrow \neg\Downarrow)$,
- $\Downarrow \wedge \mathsf{X}\Uparrow$,
- $\mathsf{G}(in \wedge \mathsf{X}\, in \wedge \Uparrow \rightarrow \mathsf{X}\Uparrow)$ ("we stay on upward sequence"),
- $\mathsf{G}(in \wedge \mathsf{X}\, in \wedge \Downarrow \rightarrow \mathsf{X}\Downarrow)$ ("we stay on downward sequence"),
- $\mathsf{G}(in \wedge \mathsf{X}\, out \wedge \Uparrow \rightarrow (\mathsf{X}\Downarrow \wedge \mathsf{X}\mathsf{X}\Downarrow))$ ("we pass from upward to downward sequence"),

- $\mathsf{G}(in \wedge \mathsf{X}\, out \wedge \Downarrow \rightarrow (\mathsf{X}\Uparrow \wedge \mathsf{X}\mathsf{X}\Uparrow))$ ("we pass from downward to upward sequence").

The only structure (built over $\{in, out, \Uparrow, \Downarrow\}$) satisfying SNAKE $\wedge$ DIRECTION is

$$\{in, \Downarrow\} \cdot \{out, \Uparrow\} \cdot \{in, \Uparrow\}^2 \cdot \{out, \Downarrow\} \cdot \{in, \Downarrow\}^3 \cdot \{out, \Uparrow\} \cdot \{in, \Uparrow\}^4 \cdots$$

This structure encodes the path through $\mathbb{N}^2$ described in Figure 1.11. The path allows to access to adjacent states as follows:

- in a state $\{in, \Uparrow\}$, we access to the up and right neighbours with the help of L_0 and L_1, respectively,
- in a state $\{in, \Downarrow\}$, we access to the up and right neighbours with the help of L_1 and L_0, respectively,

We write CONSTRAINTS to denote the conjunction of following formulae that express color constraints for adjacent dominoes:

- $\mathsf{G}(in \wedge \Uparrow \rightarrow (\bigwedge_{1 \leq i \leq n} right_i \rightarrow \mathrm{L}_1(in, out, left_i)))$,
- $\mathsf{G}(in \wedge \Uparrow \rightarrow (\bigwedge_{1 \leq i \leq n} up_i \rightarrow \mathrm{L}_0(in, out, down_i)))$,
- $\mathsf{G}(in \wedge \Downarrow \rightarrow (\bigwedge_{1 \leq i \leq n} right_i \rightarrow \mathrm{L}_0(in, out, left_i)))$,
- $\mathsf{G}(in \wedge \Downarrow \rightarrow (\bigwedge_{1 \leq i \leq n} up_i \rightarrow \mathrm{L}_1(in, out, down_i)))$.

We write REC to denote the formula that states that colour 1 occurs infinitely often in the first column:

$$\mathsf{G}\,\mathsf{F}\left(\mathsf{X}(out \wedge \Downarrow) \wedge \bigvee_{s \in \mathrm{Sides}} s_1\right) \vee \mathsf{G}\,\mathsf{F}\left(out \wedge \Downarrow \wedge \mathsf{X} \bigvee_{s \in \mathrm{Sides}} s_1\right)$$

The domino game Dom can pave $\mathbb{N}^2$ with colour 1 occurring infinitely often on the first column if and only if PAVE$\wedge$SNAKE$\wedge$DIRECTION$\wedge$CONSTRAINTS$\wedge$REC is satisfiable in PC[L_1]. □

Satisfiability for PC[L_1] is therefore Σ^1_1-hard and is not recursively enumerable. The proof of Proposition 1.14 is inspired from the undecidability of propositional dynamic logic (PDL) augmented with the context-free language $\{a_1^k a_2 a_1^k \mid k \geq 0\}$ (see for example [23, chapter 9] for more details). For formal verification, the following result is more meaningful since it illustrates the strength of adding a single context-free language.

Corollary 1.15. *The model-checking problem for* PC[L_1] *is undecidable.*

Proof. It is indeed simple to reduce satisfiability for PC[L_1] to model-checking for PC[L_1]. Let φ be a formula built over the propositional variables $p_1, \ldots, p_n$. We write $\mathcal{M}_n = \langle W, R, \lambda \rangle$ to denote the complete Kripke structure such that $W = 2^{\{p_1, \ldots, p_n\}}$, $R = W \times W$ and λ is the identity. Then φ is valid if and only if for all $s \in W$, we have $\mathcal{M}_n, s \models_\forall \varphi$. Since PC[$\mathrm{L}_1$] is closed under negations, by Proposition 1.14, the validity problem for PC[L_1] is also undecidable. Hence, the model-checking problem for PC[L_1] is undecidable. □

Surprisingly, a similar undecidability result holds for a specific context-free language that can be recognized by a visibly pushdown automaton (VPA) [22] as shown below. For instance, L_1 cannot be recognized by a VPA. Let L_2 be the context-free language $\{a_1^k a_2 a_3^k a_4 \mid k \geq 0\}$ built over the alphabet $\Sigma = \{a_1, \ldots, a_4\}$. This language can be easily defined by a VPA. A very nice feature of the class of VPA is that it defines context-free languages that are closed under Boolean operations. Moreover, PDL generalized to programs defined by VPA is still decidable [24] (generalizing for instance [57]). Unlike this extension of PDL, we have the following undecidability results.

Corollary 1.16. *Satisfiability and model-checking for* PC[L_2] *are undecidable.*

Indeed, the temporal operator defined with the language L_2 can easily express the temporal operator defined with L_1 since $L_1(\varphi_1, \varphi_2, \varphi_3) \equiv L_2(\varphi_1, \varphi_2, \varphi_1, \varphi_3)$. By contrast, the characterization of decidable positive fragments of PC[CF] (without negation) is still open (the above undecidability proofs use negation in an essential way).

1.5. Concluding remarks

In this chapter, we have presented two distinct aspects of the use of automata theory for the verification of computer systems. A translation from LTL formulae into Büchi automata has been defined with the main advantage to produce simple automata for simple formulae. This follows the automata-based approach advocated in [9] with simplicity and pedagogical requirements from [12, 14]. We believe that this is an adequate translation to be taught to students. The second use of automata is related to automata-based temporal operators generalizing the more standard temporal connectives such as the next operator or the until operator. After recalling the standard results about the operators defined from regular languages, we have explained how a restricted addition of operators defined from context-free languages can easily lead to undecidability. For instance, we show why model-checking for propositional calculus augmented with a simple CF language recognized by a VPA [22] is highly undecidable.

References

[1] P. Blackburn, M. de Rijke, and Y. Venema, *Modal Logic*. (Cambridge University Press, 2001).
[2] N. Rescher and A. Urquhart, *Temporal Logic*. (Springer-Verlag, 1971).
[3] A. Pnueli. The temporal logic of programs. In *FOCS'77*, pp. 46–57. IEEE, (1977).
[4] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *FOCS'82*, pp. 337–351. IEEE, (1982).
[5] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications: A practical approach. In *POPL'83*, pp. 117–126. ACM, (1983).

[6] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. (MIT Press, 2000).
[7] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B, Formal models and semantics*, pp. 133–191. Elsevier, (1990).
[8] R. Büchi. On a decision method in restricted second-order arithmetic. In *International Congress on Logic, Method and Philosophical Science'60*, pp. 1–11, (1962).
[9] M. Vardi and P. Wolper, Reasoning about infinite computations, *Information and Computation*. **115**, 1–37, (1994).
[10] D. Harel, O. Kupferman, and M. Vardi, On the complexity of verifying concurrent transition systems, *Information and Computation*. **173**(2), 143–161, (2002).
[11] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. In *CAV'90*, vol. 531, *Lecture Notes in Computer Science*, pp. 233–242. Springer, (1990).
[12] R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing and Verification XV*, pp. 3–18. Chapman & Hall, (1995).
[13] M. Daniele, F. Giunchiglia, and M. Vardi. Improved automata generation for linear temporal logic. In *CAV'99*, vol. 1633, *Lecture Notes in Computer Science*, pp. 249–260. Springer, (1998).
[14] J.-M. Couvreur. On-the-fly verification of linear temporal logic. In *FM'99*, vol. 1708, *Lecture Notes in Computer Science*, pp. 253–271. Springer, (1999).
[15] P. Wolper, The tableau method for temporal logic: An overview, *Logique et Analyse*. **110–111**, 119–136, (1985).
[16] P. Wolper. Constructing automata from temporal logic formulas: A tutorial. In *European Educational Forum: School on Formal Methods and Performance Analysis*, vol. 2090, *Lecture Notes in Computer Science*, pp. 261–277. Springer, (2000).
[17] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *CAV'01*, vol. 2102, *Lecture Notes in Computer Science*, pp. 53–65. Springer, (2001).
[18] D. Giannakopoulou and F. Lerda. From states to transitions: improving translation of LTL formulae to Büchi automata. In *FORTE'02*, vol. 2529, *Lecture Notes in Computer Science*, pp. 308–326. Springer, (2002).
[19] J. Couvreur, A. Duret-Lutz, and D. Poitrenaud. On-the-fly emptiness checks for generalized Büchi automata. In *SPIN'05*, vol. 3639, *Lecture Notes in Computer Science*, pp. 169–184. Springer, (2005).
[20] H. Tauriainen. *Automata and Linear Temporal Logic: Translations with Transition-based Acceptance*. PhD thesis, Helsinki University of Technology, (2006).
[21] P. Wolper, Temporal logic can be more expressive, *Information and Computation*. **56**, 72–99, (1983).
[22] R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC'04*, pp. 202–211. ACM Press, (2004).
[23] D. Harel, D. Kozen, and J. Tiuryn, *Dynamic Logic*. (MIT Press, 2000).
[24] C. Löding and O. Serre. Propositional dynamic logic with recursive programs. In *FOSSACS'06*, vol. 3921, *Lecture Notes in Computer Science*, pp. 292–306. Springer, (2006).
[25] A. Prior, *Past, Present and Future*. (Oxford University Press, 1967).
[26] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *7th Annual ACM Symposium on Principles of Programming Languages*, pp. 163–173. ACM Press, (1980).
[27] J. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, USA, (1968).
[28] Z. Manna and A. Pnueli. The modal logic of programs. In *ICALP'79*, vol. 71, *Lecture*

Notes in Computer Science, pp. 385–409. Springer, (1979).
[29] A. Pnueli. The temporal semantics of concurrent programs. In *International Symposium on Semantics of Concurrent Computation 1979*, vol. 70, *Lecture Notes in Computer Science*, pp. 1–20. Springer, (1979).
[30] C. Eisner and D. Fisman, *A Practical Introduction to PSL*. (Springer, 2006).
[31] G. Holzmann, The model checker SPIN, *IEEE Transactions on Software Engineering*. **23**(5), 279–295, (1997).
[32] K. McMillan, *Symbolic Model Checking*. (Kluwer Academic Publishers, 1993).
[33] E. Clarke and A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Worshop on Logic of Programs*, vol. 131, *Lecture Notes in Computer Science*, pp. 52–71. Springer, (1981).
[34] A. Emerson and J. Halpern, "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic, *Journal of the Association for Computing Machinery*. **33**, 151–178, (1986). Preliminary version in POPL'83, pp. 127–140.
[35] A. Sistla and E. Clarke, The complexity of propositional linear temporal logic, *Journal of the Association for Computing Machinery*. **32**(3), 733–749, (1985).
[36] S. Demri and P. Schnoebelen, The complexity of propositional linear temporal logics in simple cases, *Information and Computation*. **174**(1), 84–103, (2002).
[37] P. Schnoebelen. The complexity of temporal logic model checking. In *Advances in Modal Logic, vol. 4, selected papers from 4th Conf. Advances in Modal Logic (AiML'2002), Sep.-Oct. 2002, Toulouse, France*, pp. 437–459. King's College Publication, (2003).
[38] E. M. Clarke, E. A. Emerson, and A. P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, *ACM Transactions on Programming Languages and Systems*. **8**(2), 244–263, (1986).
[39] E. Emerson and C. Jutla, The complexity of tree automata and logics of programs, *SIAM Journal of Computing*. **29**(1), 132–158, (2000).
[40] M. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *STOC'85*, pp. 240–251. ACM, (1985).
[41] M. Vardi. Nontraditional applications of automata theory. In *TACS'94*, vol. 789, *Lecture Notes in Computer Science*, pp. 575–597. Springer, (1994).
[42] K. Etessami, T. Wilke, and R. Schuller, Fair simulation relations, parity games, and state space reduction for Büchi automata, *SIAM Journal of Computing*. **34**(5), 11591175, (2005).
[43] A. Meyer. Weak second order theory of successor is not elementary-recursive. Technical Report MAC TM-38, MIT, (1973).
[44] L. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, Department of Electrical Engineering, MIT, (1974).
[45] K. Etessami and G. Holzmann. Optimizing Büchi automata. In *CONCUR'00*, vol. 1877, *Lecture Notes in Computer Science*, pp. 153–167. Springer, (2000).
[46] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *CAV'00*, vol. 1855, *Lecture Notes in Computer Science*, pp. 248–263. Springer, (2000).
[47] C. Papadimitriou, *Computational Complexity*. (Addison-Wesley Publishing Company, 1994).
[48] Z. Manna and A. Pnueli, *Temporal Verification of Reative Systems: Safety*. (Springer-Verlag, New York, 1995).
[49] S. Schwendimann. A new one-pass tableau calculus for PLTL. In *TABLEAUX'98*, vol. 1397, *Lecture Notes in Artificial Intelligence*, pp. 277–291. Springer, (1998).
[50] E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In *CAV'94*, vol. 818, *Lecture Notes in Computer Science*, pp. 415–427. Springer, (1994).

[51] V. Diekert and P. Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives*, vol. 2, *Texts in Logic and Games*, pp. 261–306. Amsterdam University Press, (2008).
[52] H. Straubing, *Finite Automata, Formal Logic, and Circuit Complexity*. Progress in Theoretical Computer Science, (Birkhäuser, 1994).
[53] P. Wolper, M. Vardi, and A. Sistla. Reasoning about infinite computation paths. In *FOCS'83*, pp. 185–194, (1983).
[54] M. Vardi. A temporal fixpoint calculus. In *POPL'88*, pp. 250–259. ACM, (1988).
[55] D. Kozen, *Automata and Computability*. (Springer, 1997).
[56] D. Harel, Recurring dominoes: making the highly undecidable highly understandable, *Annals of Discrete Mathematics*. **24**, 51–72, (1985).
[57] D. Harel and E. Singerman, More on nonregular PDL: Finite models and Fibonacci-like programs, *Information and Computation*. **128**, 109–118, (1996).

Index

ω-regular language, 28

automaton
  Büchi automaton, 11
  finite-state automaton, 27
  generalized Büchi automaton, 12
  transition-based acceptance, 12
  visibly pushdown automaton, 3, 34

Büchi automaton, 11
BA, *see* Büchi automaton

computation tree logic CTL, 9, 10
CTL, *see* computation tree logic
CTL*, 7, 25

domino game, 31

ETL, *see* extended temporal logic
extended temporal logic ETL, 27

GBA, *see* generalized Büchi automaton
generalized Büchi automaton, 12

Kripke structure, 6

language
  ω-regular, 28
  context-free, 29
  regular, 27
linear-time temporal logic LTL, 5
logic, *see* temporal logic
  monadic second-order logic MSO, 29
  propositional dynamic logic PDL, 3
LTL, *see* linear-time temporal logic

modality, 3
model-checking
  algorithm, 25
  complexity, 9, 22, 28, 29
  problem, 7, 9, 10
monadic second-order logic MSO, 29
MSO, *see* monadic second-order logic

past-time operator, 6
path quantifier, 7
PDL, *see* propositional dynamic logic
problem
  model-checking, 7, 9, 10
  recurring domino, 31
  satisfiability, 6
  universality, 30
  validity, 6, 30
propositional dynamic logic PDL, 3

recurring domino problem, 31
renaming, 25

satisfiability problem, 6

temporal logic, 3
  CTL*, 7, 25
  computation tree logic CTL, 9, 10
  extended temporal logic ETL, 27
  linear-time temporal logic LTL, 5
transition-based acceptance, 12

universality problem, 30

validity problem, 6
visibly pushdown automaton, 3, 34
VPA, *see* visibly pushdown automaton